Courant Computer Science Report No.21

October 1980

Circle Graphs

Mark A. Buckingham

Courant Institute of
Mathematical Sciences

Computer Science Department

New York University

Report No. NSO-21 prepared under Grant No.
NSF-MCS 78-03820 from the National Science Foundation

COURANT COMPUTER SCIENCE PUBLICATIONS

COURANT COMPUTER SCIENCE NOTES

A101 ABRAHAMS, P. The PL/I Programming Language, 1979, 151 p.

C66 COCKE, J. & SCHWARTZ, J. Programming Languages and Their Compilers, 1970, 767 p.

D86 DAVIS, M. Computability, 1974, 248 p.

M72 MANACHER, G. ESPL: A Low-Level Language in the Style of Algol, 1971, 496 p.

M81 MULLISH, H. & GOLDSTEIN, M. A SETLB Primer, 1973, 201 p.

S91 SCHWARTZ, J. On Programming: An Interim Report on the SETL Project. Generalities; The SETL Language and Examples of Its Use. 1975, 675 p.

S99 SHAW, P. GYVE — A Programming Language for Protection and Control in a Concurrent Processing Environment, 1978, 668 p.

S100 SHAW, P. " Vol. 2, 1979, 600 p.

W78 WHITEHEAD, E.G., Jr. Combinatorial Algorithms, 1973, 104 p.

COURANT COMPUTER SCIENCE REPORTS

1 WARREN, H. Jr. ASL: A Proposed Variant of SETL, 1973, 326 p.

2 HOBBS, J. R. A Metalanguage for Expressing Grammatical Restrictions in Nodal Spans Parsing of Natural Language, 1974, 266 p.

3 TENENBAUM, A. Type Determination for Very High Level Languages, 1974, 171 p.

5 GEWIRTZ, W. Investigations in the Theory of Descriptive Complexity, 1974, 60 p.

6 MARKSTEIN, P. Operating System Specification Using Very High Level Dictions, 1975, 152 p.

7 GRISHMAN, R. (ed.) Directions in Artificial Intelligence: Natural Language Processing, 1975, 107 p.

8 GRISHMAN, R. A Survey of Syntactic Analysis Procedures for Natural Language, 1975, 94 p.

9 WEIMAN, CARL Scene Analysis: A Survey, 1975, 62 p.

10 RUBIN, N. A Hierarchical Technique for Mechanical Theorem Proving and Its Application to Programming Language Design, 1975, 172 p.

11 HOBBS, J.R. & ROSENSCHEIN, S.J. Making Computational Sense of Montague's Intensional Logic, 1977, 41 p.

12 DAVIS, M. & SCHWARTZ, J. Correct-Program Technology/Extensibility of Verifiers, with an Appendix by E. Deak, 1977, 146 p.

13 SEMENIUK, C. Groups with Solvable Word Problems, 1979, 77 p.

14 FABRI, J. Automatic Storage Optimization, 1979, 159 p. ..

15. LIU, S-C. & PAIGE, R. Data Structure Choice/Formal Differentiation. Two Papers on Very High Level Program Optimization, 1979, 658 p.

16 GOLDBERG, A. T. On the Complexity of the Satisfiability Problem, 1979, 85 p.

17 SCHWARTZ, J.T. & SHARIR, M. A Design for Optimizations of the Bitvectoring Class, 1979, 71 p.

18 STOLFO, S. J. Automatic Discovery of Heuristics for Nondeterministic Programs from Sample Execution Traces, 1979, 168 pp.

19 LOERINC, B. M. Computing Chromatic Polynomials from Special Families of Graphs, 1980, 111 pp.

20 TURCHIN, V. F. The Language Refal--The Theory of Compilation and Metasystem Analysis, 1980, 245 p.

21 BUCKINGHAM, M. A. Circle Graphs, 1980, 152 pp.

Notes: Available from Department LN. Prices on request.
Reports: Available from Ms. Lenora Greene. No. 1,3,6,7,8,10 available in xerox only.

COURANT INSTITUTE OF MATHEMATICAL SCIENCES
251 Mercer Street, New York, N.Y. 10012

COURANT INSTITUTE OF MATHEMATICAL SCIENCES

Computer Science NSO-21

CIRCLE GRAPHS

Mark A. Buckingham

Copyright © 1980
Courant Institute of Mathematical Sciences
All rights reserved

Report No. NSO-21 prepared under
Grant No. NSF-MCS 7803820 from
the National Science Foundation

The research for this paper was supported in part by
a grant from the National Science Foundation.
Reproduction in whole or in part is permitted
for any purpose of the United States Government.

ABSTRACT

Title: Circle Graphs

Author: Mark A. Buckingham

Advisor: Martin C. Golumbic

From a circle with chords we may derive a graph whose nodes correspond to chords and whose edges correspond to intersecting chords. Such a graph is called a circle graph. After numbering the endpoints of the chords such that two endpoints are numbered the same iff the endpoints belong to the same chord, we form a circle graph sequence by reading off these numbers going around the outside of the circle. Circle graph sequences are often used to prove properties of circle graphs.

In this dissertation we discuss many mathematical and algorithmic aspects of circle graphs. The number of different circle with chords representations that yield a chordless path is given. The property that a circle with chords is connected (that is, its derived circle graph is connected) and the property that a circle with chords has two separated chords (that is, two chords that cannot both be intersected by a third chord without the third intersecting a fourth chord) are described in terms of circle graph sequences. They are found

C.2

to be dual to one another. An incomplete forbidden subgraph characterization of circle graphs is also presented.

An important result of this dissertation is that the Berge Strong Perfect Graph Conjecture is shown to hold for the class of circle graphs. Many properties of p-critical graphs and partitionable graphs are given, most with simplified proofs. Some new results are presented and a new, very simple proof of the Berge Conjecture for $K_{1,3}$-free graphs is put forward.

Very efficient algorithms for finding maximum (weighted) cliques and maximum (weighted) stable sets of the derived circle graph of a circle graph sequence are given. We find an $O(e*\log_2\omega)$ algorithm for the unweighted clique problem, an $O(\delta e)$ algorithm for the weighted clique problem and an $O(c)$ algorithm for the weighted stable set problem; where e is the number of edges in the graph, ω the maximum clique size, δ the maximum degree and c the number of occurrences of an interval being completely contained in another interval in the circle graph sequence.

Some open problems for further research are listed.

ACKNOWLEDGMENTS

I would like to express my appreciation and thanks to Professor Martin C. Golumbic who first introduced me to graph theory, challenged me in many of its fruitful areas, and saw me through this dissertation through thick and thin. His ability to distinguish between the relevant and irrelevant was a tremendous guide. And as a source of information he was invaluable.

A very, very special thanks to my wife, Lynn, for marrying me amidst promises of soon finishing this work, and then waiting patiently another full year to see it actually happen.

TABLE OF CONTENTS

	Page
ABSTRACT	ii
ACKNOWLEDGMENTS	iv
CHAPTER 1: INTRODUCTION	
1.1 The origins of circle graphs	1
1.2 Definitions and notations	6
CHAPTER 2: SOME BASIC PROPERTIES OF CIRCLE GRAPHS	
2.1 Circle graph sequences and overlap graphs	10
2.2 Some important examples of circle graph sequences	12
2.3 More complex examples of circle graph sequences	22
2.4 Counting paths	30
2.5 Connectivity	35
2.6 Separability	40
2.7 Recognition	48
CHAPTER 3: THE BERGE STRONG PERFECT GRAPH CONJECTURE	
3.1 p-Critical and partitionable graphs	56
3.2 Properties of partitionable graphs	60
3.3 Circle graphs	75
3.4 $K_{1,3}$-free graphs	85

CHAPTER 4: ALGORITHMIC ASPECTS

4.1 Sorting on a system of parallel stacks 91

4.2 Generating edges 101

4.3 An efficient clique algorithm 105

4.4 An efficient stable set algorithm 119

CHAPTER 5: OPEN PROBLEMS 125

APPENDIX

A.1 Introduction to the BALM programming language 127

A.2 Programs to find π and s, given the other, such that $H(\pi) = G(s)$ + isolated nodes 129

A.3 Programs to find the edges of G(s), C(s) and D(s) 131

A.4 Programs to find the maximum (weighted) clique 133

A.5 Programs to find the maximum (weighted) stable set 137

BIBLIOGRAPHY 140

CHAPTER 1: INTRODUCTION

1.1 THE ORIGINS OF CIRCLE GRAPHS

Graphs have long been used as a modeling tool in many disciplines. In this dissertation we look at a particular class of graphs called circle graphs. A circle graph is a graph derived from a collection of intersecting chords of a circle. The nodes of a circle graph correspond to the chords of the circle and the edges correspond to the intersection of two chords.

Such a graph arises when one considers a connection board with connections allowed on every side.

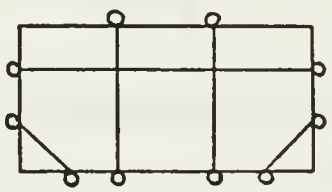


Figure 1.1.1 A connection board.

The board could model an actual electric circuit diagram where current is to run uninterrupted along a circuit. Or the board could represent a collection of gas lines that are to be buried in the ground, and we need to determine in what order the lines are laid. Some connections will cause the intervening paths to

cross and therefore must be placed in different planes. One problem to be considered is: What is the minimum number of planes that are needed to realize a given board, and on which plane is each circuit to be placed?

Another area in which a circle graph arises is the problem of sorting a given permutation on a system of parallel stacks.

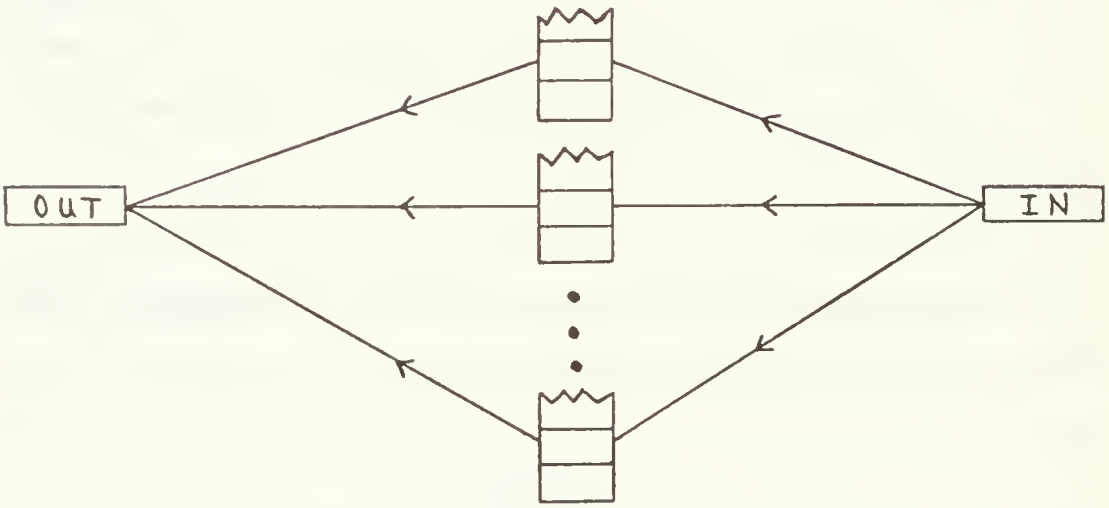


Figure 1.1.2 Sorting using a system of parallel stacks.

The system could be considered as a train yard and the stacks as sidings, and the goal is to rearrange an incoming train. Likewise, the system could be considered as an electrical circuit where the components act as stacks and the goal is to rearrange a stream of input. To achieve certain arrangements some input values must be placed on different stacks. In such a system one problem to be raised is: What is the minimum

number of stacks that are needed to sort a given permutation, and on which stack must each value be placed?

It is easy to see how the connection board can be represented by a circle with chords: Just round out the sides of the connection board, leaving the circuits straight. In the case of the permutation sorting system, this is not so easy to see. Even and Itai [1971] showed the sorting system to be representable as a circle with chords, where intersecting chords correspond to two input values needing to be placed on different stacks. Hence, the two previous problems can be viewed as a coloring problem on their respective circles with chords: Using a minimum number of colors, assign a color to each chord such that no two intersecting chords have the same color. Each color corresponds to the plane or stack on which the circuit path or input value is to be placed.

Circle graphs also relate to a number of other intersection derived graphs. Let C_1= the class of permutation graphs, graphs derived from the intersecting lines of a matching diagram, C_2= the class of circle graphs, C_3= the class of graphs derived from the intersecting lines of a plane and C_4= the class of graphs derived from the intersecting curves of a plane. We have the following relationships (Ehrlich, Even and Tarjan [1976]):

$$C_1 \subsetneq C_2 \subsetneq C_3 \subsetneq C_4 \subsetneq \text{the class of all graphs.}$$

On C_1 the usually NP-complete problems of finding a maximum

weighted clique, maximum weighted stable set, minimum coloring and minimum clique covering are all polynomial. None of these, of course, are known to be polynomial on the class of all graphs. Therefore, somewhere in this progression of classes of graphs each of these problems goes from polynomial to NP-complete. Where does the class of circle graphs stand with respect to these problems?

Many other graph theoretic problems can and will be applied to the class of circle graphs: What are the forbidden subgraphs of the class of circle graphs, is there a polynomial algorithm for determining circle graphs, and is the Berge Strong Perfect Graph Conjecture true for circle graphs?

In chapter 1, circle graphs are introduced, and definitions and notations are given.

In chapter 2, we discuss circle graphs which have a unique circle with chords representation and also show many graphs that are not circle graphs. The number of different circle with chords representations that yield a chordless path is given. The property that a circle with chords is connected (that is, its derived circle graph is connected) and the property that a circle with chords has two separated chords (that is, two chords that cannot both be intersected by a third chord without the third intersecting a fourth chord) are described in terms of circle graph sequences. They are found

to be dual to one another. An incomplete forbidden subgraph characterization of circle graphs is also presented.

In chapter 3, we show that the Berge Strong Perfect Graph Conjecture holds for the class of circle graphs. Many properties of p-critical graphs and partitionable graphs are given, most with simplified proofs. Some new results are presented and a new, very simple proof of the Berge Conjecture for $K_{1,3}$-free graphs is put forward.

In chapter 4, very efficient algorithms for finding maximum (weighted) cliques and maximum (weighted) stable sets of the derived circle graph of a circle graph sequence are given. We find an $O(e*\log_2\omega)$ algorithm for the unweighted clique problem, an $O(\delta e)$ algorithm for the weighted clique problem and an $O(c)$ algorithm for the weighted stable set problem, the unweighted case is just a subcase; where e is the number of edges in the graph, ω the maximum clique size, δ the maximum degree and c the number of occurrences of an interval being completely contained in another interval in the circle graph sequence.

In chapter 5, some open problems for further research are listed.

1.2 DEFINITIONS AND NOTATIONS

A graph is a finite set of undefined objects called nodes, and a set of unordered pairs of these nodes called edges. We allow no self-loops or multiple edges. A subgraph, of a graph having nodes V and edges E, has nodes V' contained in V and edges E' contained in E such that E' is contained in V' x V', and i,j in V' and (i,j) in E imply (i,j) is in E'. That is, our notion of a subgraph is the node induced subgraph. A spanning subgraph, of a graph having nodes V and edges E, has nodes V' equal to V and edges E' contained in E.

The adjacency set of a node i, adj(i), consists of those nodes j such that (i,j) is an edge. The neighborhood of a node i, N(i), is adj(i) U i. The degree of a node i is $\|adj(i)\|$.

Two nodes i and j are adjacent if (i,j) is an edge. A path is a subgraph where the nodes can be ordered such that consecutive nodes in this ordering are adjacent. A chordless path is a path having no other edges but the adjoining edges. P_n will denote the chordless path on n nodes. A cycle is a path plus an edge between the first and last node. A chordless cycle is a chordless path plus an edge between the first and last node. C_n will denote the chordless cycle on n nodes.

P_n^k will denote the graph of n nodes labeled 1,2,...,n and edges (i,j) iff $\|i-j\| < k$. That is, P_n^k is a path of n nodes with edges between the nodes that are no more than k apart. Clearly, $P_n^1 = P_n$. Similarly C_n^k will denote the graph of n nodes labeled 1,2,...,n and edges (i,j) iff $\|i-j\| < k$, modulo n. C_n^k is a cycle of n nodes with edges between the nodes that are no more than k apart. Again, $C_n^1 = C_n$.

G+x will denote the graph G with an additional node adjacent to all nodes of G. G-S will denote the resulting graph after removing all nodes of the set S and edges involving these nodes. G-x will be equivalent to G-{x}.

A connected component is a maximal subgraph where there is a path between each pair of nodes. A graph is connected if it is comprised of one connected component. Two subgraphs are connected if a node of one is adjacent to a node of the other. A biconnected component is a maximal subgraph where for each pair of nodes there is a cycle containing them or, equivalently, a connected subgraph where the removal of any node still results in a connected subgraph.

A clique is a subgraph where every pair of nodes is adjacent. K_n will denote a clique on n nodes. A stable set is a subgraph having no edges. S_n will denote a stable set on n nodes.

The complement of a graph is a graph having the same set of nodes and having edges (i,j) iff (i,j) is not an edge of the original graph.

We assume, without loss of generality (WLOG), that the chords of a circle have distinct endpoints, and that the intervals of a line also have no endpoints in common. The derived graph of a circle with chords has one node associated with each chord and an edge for each pair of nodes whose corresponding chords intersect.

A circle graph sequence, besides being derived from the endpoints of the chords of a circle, or from the endpoints of the intervals of a line, can be more abstractly defined as a finite sequence of numbers in which every number occurs exactly twice.

Given a circle graph sequence we can list the sequence around a circle and connect each pair of matching numbers. The result is called the underlying or corresponding circle with chords.

In a circle graph sequence the two occurrences of a given number i will be referred to as the left and right endpoints of interval i. The interior or inside of an interval i of a circle graph sequence is all those elements of the sequence between the endpoints of i, and the exterior or outside is

those that are not inside the interval. An interval i contains an element if that element is in the interior of i. Two intervals i,j of the sequence overlap if interval i contains exactly one endpoint of j. The derived graph of a circle graph sequence has one node corresponding to each interval and an edge for each pair of nodes whose corresponding intervals overlap. This is equivalent to the derived graph of the underlying circle with chords.

A graph is representable, or constructable, by a circle graph sequence, or a circle with chords, if there exists a circle graph sequence, or a circle with chords, that derives the graph.

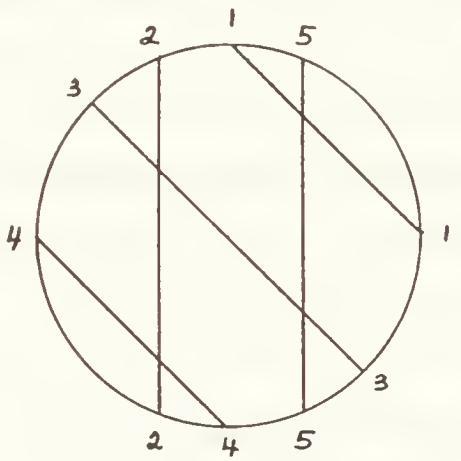
Let S and S' be arbitrary sequences. S' is a subsequence of S if the elements of S' are a subset of the elements of S, and are listed in the same order as in S.

O(f(x)), where x is a non-negative vector, is a function of x such that there exists positive constants c_1, c_2 where $c_1 f(x) < O(f(x)) < c_2 f(x)$, for all but a finite number of x. A problem is polynomial if there exists an algorithm that solves it that has a time bound of O(p(x)) for some polynomial p. Two problems are polynomially equivalent if an instance of one can be transformed into an instance of the other, and vice versa, in polynomial time.

CHAPTER 2: SOME BASIC PROPERTIES OF CIRCLE GRAPHS

2.1 CIRCLE GRAPH SEQUENCES AND OVERLAP GRAPHS

A circle with chords has a very simple representation. Number the endpoints of the chords such that two endpoints are numbered the same iff the endpoints belong to the same chord. From this, form a sequence by reading off these numbers going around the outside of the circle. Call this sequence a circle graph sequence.



[1 2 3 4 2 4 5 3 1 5]

Figure 2.1.1 A circle with chords and a corresponding circle graph sequence.

It is clear that from this sequence the original circle with chords can be uniquely reconstructed. In fact, from this

representation the corresponding circle graph itself can be constructed in $O(|V| + |E|)$ time (see section 4.2).

Again, consider a circle with chords, this time project these chords from a given point of the circle onto a line tangent to the circle. Under this projection the chords become intervals, and two chords of the circle intersect iff the corresponding intervals properly overlap, that is, intersect without one being contained in the other. A graph that is derived from properly overlapping intervals on a line is called an overlap graph. Hence we have the following: A graph is a circle graph iff it is an overlap graph.

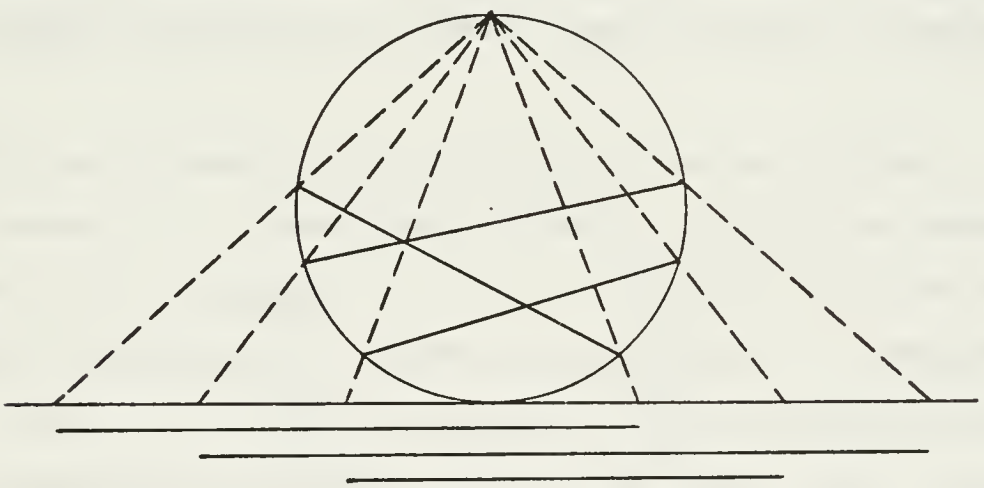


Figure 2.1.2 A circle with chords and corresponding intervals.

A circle graph sequence corresponds to a numbering of the endpoints of the intervals with matching numbers at the endpoints of each interval, and thus is a representation of intervals on a line.

2.2 SOME IMPORTANT EXAMPLES OF CIRCLE GRAPH SEQUENCES

Given a circle with chords there are clearly many corresponding circle graph sequences. A simple renumbering, starting the numbering at a different endpoint, or even reading off the numbers in the opposite direction generally yields a different sequence. Label these operations as renumber, rotation and reflection, respectively. Two circle graph sequences are equivalent if they differ by only a combination of renumber, rotation and reflection. That is, if the underlying circle with chords is at most rotated around or flipped over. In essence, all circle graph sequences of a particular circle with chords are equivalent; not being equivalent implies different underlying circles with chords.

Consider the set of all circle graph sequences for a given circle with chords. Let the smallest sequence, with respect to lexicographic ordering, be called the canonical circle graph sequence for the circle with chords. The canonical sequence can be easily calculated from any sequence of a particular circle with chords as follows: write out all 4n equivalent sequences by selecting different endpoints on the circle as the starting point, and include all reversals (reflections); then renumber the sequences so that the left endpoint of interval 1 occurs first, 2 second, etc; by observation choose the smallest. This quick and dirty algorithm takes $O(n^2)$ time, and $O(n)$ space if only the best sequence is kept as the sequences

are generated. Many shortcuts exist as the reader will find out for himself after a little practice, such as a lone [... x x ...] pattern will have [1 1 ...] as its canonical equivalent.

LEMMA 2.2.1 [1 1 2 2 ... n n] is the smallest canonical sequence on n intervals.

Proof: [1 1 2 2 ... n n] is the smallest of all circle graph sequences on n intervals and hence must be canonical. Therefore it is the smallest of all canonical sequences. ∎

The sequence of Lemma 2.2.1 derives S_n with the corresponding circle with chords having chords going 'around' the circle (see Figure 2.2.1).

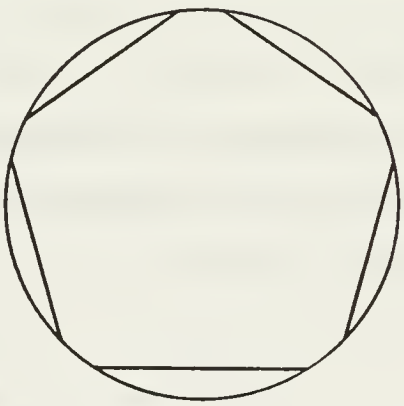


Figure 2.2.1 [1 1 2 2 3 3 4 4 5 5] derives S_5.

Let s_i denote the ith element of sequence S.

LEMMA 2.2.2 [1 2 1 3 2 ... n n-1 n] is the smallest canonical sequence on n intervals that derives a connected graph.

Proof: Let S= [1 2 1 3 2 ... n n-1 n]. Clearly S is canonical and derives a connected graph. Assume there is a smaller sequence T that derives a connected graph. Thus there exists k, $1<k<2n$, such that $s_i=t_i$ for $i<k$ and $s_k>t_k$.

Case i) If s_k is a left endpoint, then $[t_1\ t_2\ \ldots\ t_{k-1}]$ contains $1,2,\ldots,s_k-2$ twice and s_k-1 once. Thus $t_k=s_k-1$. But then $[t_1\ t_2\ \ldots\ t_k]$ contains $1,2,\ldots,s_k-1$ twice and no others, and hence no interval of it overlaps one from $[t_{k+1}\ t_{k+2}\ \ldots\ t_{2n}]$. ++

Case ii) If s_k is a right endpoint, then $[t_1\ t_2\ \ldots\ t_{k-1}]$ contains $1,2,\ldots,s_k-1$ twice and s_k, s_k+1 once. But then $t_k>s_k$. ++ ∎

The sequence of Lemma 2.2.2 derives P_n with the corresponding circle with chords having chords going 'around' the circle (see Figure 2.2.2).

Figure 2.2.2 [1 2 1 3 2 4 3 5 4 5] derives P_5.

LEMMA 2.2.3 [1 2 ... n 1 2 ... n] is the largest canonical sequence on n intervals.

Proof: Let S= [1 2 ... n 1 2 ... n]. Clearly S is canonical. Assume there is a larger canonical sequence T. Thus there exists k, $1<k<2n$, such that $s_i=t_i$ for $i<k$ and $s_k<t_k$. Case i) If $k\leq n$, then exchanging the numbering of intervals t_k and k yields a smaller sequence equivalent to T. →← Case ii) If $k>n$, then the left endpoint of interval t_k is in position t_k. But then the number of sequence elements inside interval t_k is $k-t_k-1 < k-s_k-1 = k-(k-n)-1 = n-1$. Thus rotating T such that the left endpoint of t_k is first yields a sequence equivalent to [1 2 ... $k-t_k$ 1 ...] which is less than S. →← ∎

The sequence of Lemma 2.2.3 derives K_n with the corresponding circle with chords being a 'star' (see Figure 2.2.3).

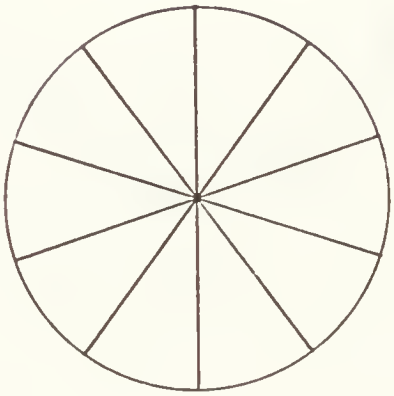


Figure 2.2.3 [1 2 3 4 5 1 2 3 4 5] derives K_5.

LEMMA 2.2.4 [1 2 ... n 1 2 ... n] is the unique canonical sequence that derives K_n.

Proof: Clearly the sequence is canonical and derives K_n. To show uniqueness we will use induction on n. For n=1 it is true. Assume n>1.

Let S be a canonical sequence that derives K_n. Form S' by removing interval n from S. By induction S' is equivalent to [1 2 ... n-1 1 2 ... n-1]. Adding n back again to this yields [1 2 ... j n j+1 j+2 ... n-1 1 2 ... j n j+1 j+2 ... n-1] for some j<n-1 or [n 1 2 ... n-1 n 1 2 ... n-1]. But these are equivalent to [1 2 ... n 1 2 ... n]. ∎

LEMMA 2.2.5 [1 2 1 2], [1 2 1 3 2 3] and [1 2 1 3 2 4 3 4] are the unique canonical sequences that derive P_2, P_3 and P_4, respectively.

Proof: Clearly each sequence is canonical and derives its respective graph.

i) $P_2 = K_2$, hence by Lemma 2.2.4 [1 2 1 2] is unique.

ii) $P_3 = P_2$ with a node attached to one of the nodes of degree 1. WLOG overlap interval 3 with 2 in [1 2 1 2]. There are three possibilities: [1 3 2 3 1 2], [3 1 2 1 3 2] and [1 2 1 3 2 3]. Each of these is equivalent to [1 2 1 3 2 3].

iii) $P_4 = P_3$ with a node attached to one of the nodes of degree 1. WLOG overlap interval 4 with 3 in [1 2 1 3 2 3]. Again, there are only three possibilities: [1 2 1 4 3 4 2 3], [4 1 2 1 3 2 4 3] and [1 2 1 3 2 4 3 4]. And again each of these is equivalent to [1 2 1 3 2 4 3 4]. ∎

LEMMA 2.2.6 P_5 is derived by two canonical sequences: [1 2 1 3 2 4 3 5 4 5] and [1 2 1 3 2 4 5 4 3 5].

Proof: Clearly each derives P_5 and is canonical. Begin with P_4, [1 2 1 3 2 4 3 4], and WLOG overlap interval 5 with 4. There are three possibilities: [1 2 1 3 2 5 4 5 3 4], [5 1 2 1 3 2 4 3 5 4] and [1 2 1 3 2 4 3 5 4 5]. The last two are equivalent to [1 2 1 3 2 4 3 5 4 5] and the first is equivalent to [1 2 1 3 2 4 5 4 3 5]. ∎

The difference between the two sequences of Lemma 2.2.6 can be more clearly seen in their underlying circles with chords (see Figure 2.2.4).

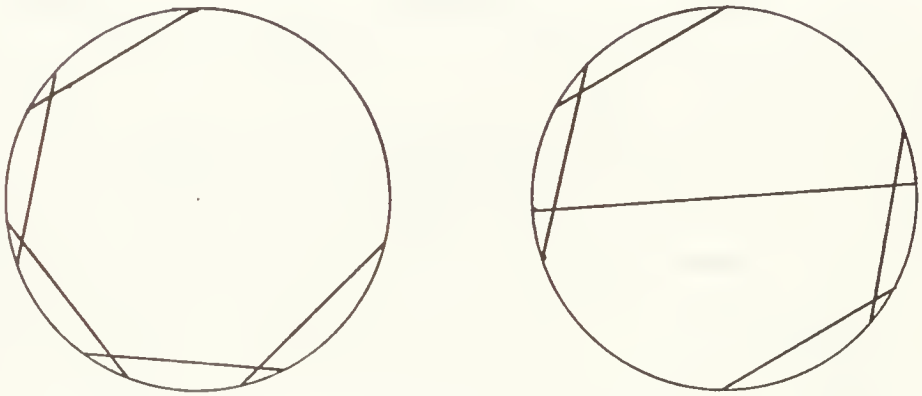


Figure 2.2.4 [1 2 1 3 2 4 3 5 4 5] and [1 2 1 3 2 4 5 4 3 5].

Recall that S' is a subsequence of S if the elements of S' are a subset of the elements of S and are listed in the same order as in S. A distinctive difference between [1 2 1 3 2 4 3 5 4 5] and [1 2 1 3 2 4 5 4 3 5] is that [1 1 3 4 4 3] is a subsequence of the latter and not the former. In fact, [1 1 3 4 4 3] is not even equivalent to any subsequence of the former.

Two intervals are <u>separated</u> if they cannot both be overlapped by a third interval without the third overlapping a fourth interval. In the sequence [1 1 2 3 3 2], intervals 1 and 3 are separated since any interval that would overlap them both must also overlap 2. This notion will be fully examined in section 2.6. But for now consider a circle graph sequence that derives C_n, the chordless cycle of n nodes. The removal

of any interval results in a sequence that derives P_{n-1}, where the two intervals corresponding to the endpoints are not separated: The interval that was removed overlaps these two without overlapping any other interval.

LEMMA 2.2.7 [1 2 1 3 2 ... n n-1 n] is the unique canonical sequence that derives P_n where the two intervals corresponding to the endpoints are not separated.

Proof: Clearly [1 2 1 3 2 ... n n-1 n] is canonical, derives P_n and the two intervals corresponding to the endpoints, 1 and n, are not separated as evidenced by interval w: [1 w 2 1 3 2 ... n n-1 w n]. To show uniqueness we will use induction on n. By Lemma 2.2.5, for n= 2, 3 or 4 it is true. Assume n>4, and observe S, a circle graph sequence that derives P_n where the two intervals corresponding to the endpoints are not separated. We will begin by showing that no two intervals of S are separated.

Since S derives P_n, [1 2 1 2 n n] is equivalent to some subsequence of S where 1 and n are the intervals corresponding to the endpoints and 2 is the adjacency set of 1. Since 1 and n are not separated there is a placing of interval w in S such that w overlaps only 1 and n. Thus S+w contains a subsequence equivalent to [1 w 2 1 2 n w n] with 1, 2 and n as previously defined. Since 1 and w overlap no other interval, all endpoints between the left endpoint of w and the right endpoint

of 1 are paired, except 2. Thus moving the left endpoint of w so that it immediately follows the right endpoint of 1 results in w overlapping only 2 and n. Therefore 2 and n are not separated in S. Remove interval 1 and iteratively continue. Since 1 and n are symmetric cases we have the conclusion that no two intervals of S are separated.

Form S' by removing an interval corresponding to an endpoint from S. We now know that S' is a sequence that derives P_{n-1} where the two intervals corresponding to the endpoints are not separated. By induction, S' is equivalent to: [1 2 1 3 2 ... n-1 n-2 n-1]. Let' s now reconstruct S by adding interval n back in. There are three possibilities:

i) [1 2 1 3 2 ... n-2 n-3 n n-1 n n-2 n-1],

ii) [n 1 2 1 3 2 ... n-1 n-2 n n-1] and

iii) [1 2 1 3 2 ... n-1 n-2 n n-1 n].

We can exlude i) because of the [1 1 n-2 n n n-2] subsequence which implies that the two intervals corresponding to the endpoints, 1 and n, are separated. ii) and iii) are both equivalent to [1 2 1 3 2 ... n n-1 n]. ∎

THEOREM 2.2.8 [1 2 3 1 4 3 5 4 ... n n-1 2 n] is the unique canonical sequence that derives C_n.

Proof: Observe that [1 2 3 1 4 3 5 4 ... n n-1 2 n] is equivalent to [1 n 2 1 3 2 ... n-1 n-2 n n-1]. Clearly this derives C_n and the former is its canonical equivalent.

Let S be a circle graph sequence that derives C_n. The removal of any interval results in a sequence that derives P_{n-1} where the two intervals corresponding to the endpoints are not separated. By Lemma 2.2.7 this sequence is equivalent to [1 2 1 3 2 ... n-1 n-2 n-1]. To reconstruct S, interval n must be added to overlap only 1 and n-1. This can be done in only one way, resulting in [1 n 2 1 3 2 ... n-1 n-2 n n-1]. ∎

C_n is uniquely represented by a circle with chords where the chords go 'around' the circle (see Figure 2.2.5).

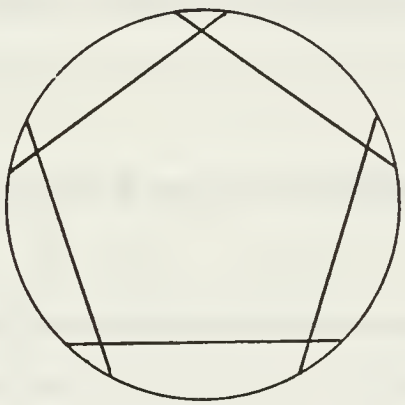


Figure 2.2.5 [1 5 2 1 3 2 4 3 5 4] derives C_5.

COROLLARY 2.2.9 C_5+x, the wheel with five sides, is not a circle graph (see Figure 2.2.6).

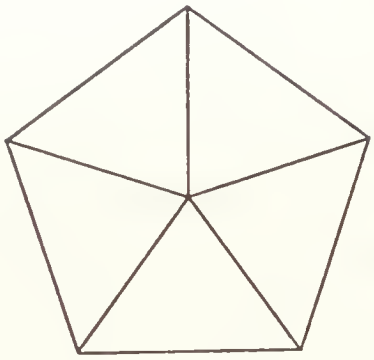


Figure 2.2.6 The wheel with five sides.

In section 2.7 we will see a generalization of this and show other graphs that are not circle graphs.

2.3 MORE COMPLEX EXAMPLES OF CIRCLE GRAPH SEQUENCES

In this section we shall show that P_n^k and C_n^k, for $k>2$, are uniquely represented by a circle with chords. Recall that P_n, for $n>5$, is not uniquely represented, although C_n is.

LEMMA 2.3.1 [1 2 ... k k+1 1 k+2 2 ... n n-k n-k+1 n-k+2 ... n] is the unique canonical sequence that derives P_n^k, for $2<k<n-1$. (Note: For $k>n-1$, $P_n^k = K_n$.)

Proof: Clearly the sequence is canonical and derives P_n^k. Let k be an arbitrary integer > 2. We shall proceed by induction on n. The initial case of n= k+1 is true by Lemma

2.2.4 since $P^k_{k+1} = K_{k+1}$. For n= k+2, $P^k_{k+2} = K_{k+1}$ plus one node, k+2, that is adjacent to all nodes but one of K_{k+1}. Observe the unique canonical sequence that derives K_{k+1}: [1 2 ... k+1 1 2 ... k+1]. WLOG assume the interval not overlapped by k+2 is 1. There are only three possibilities for adding interval k+2 such that it overlaps 2,3,...,k+1 and not 1:

i) [k+2 1 2 ... k+1 1 k+2 2 3 ... k+1],

ii) [1 2 ... k+1 1 k+2 2 3 ... k+1 k+2] and

iii) [1 k+2 2 3 ... k+1 k+2 1 2 3 ... k+1].

By interchanging the numbering of 1 and k+2 in iii), we see that i) and iii) are equivalent, and by rotating the first element of i) to the end we obtain ii). Thus all three sequences are equivalent and ii) is the desired canonical sequence.

Assume n>k+3. $P^k_n = P^k_{n-1}$ plus one node n, where adj(n)= {n-1} U adj(n-1) - one node of maximum degree from adj(n-1). By induction [1 2 ... k+1 1 k+2 2 ... n-2 n-k-2 n-1 n-k-1 n-k n-k+1 ... n-1] is the unique canonical sequence deriving P^k_{n-1}. Since 1,2,...,n-k-2 are not in adj(n-1), n cannot overlap any of them. Hence, except for the first position, the farthest left in the sequence that the left endpoint of n could occur is just prior to the left endpoint of n-1. There are only three possibilities for n to overlap n-1 and all but one of adj(n-1):

i) [n 1 2 ... k+1 1 k+2 2 ... n-2 n-k-2 n-1 n-k-1 n n-k n-k+1 ... n-3 n-2 n-1],

ii) [1 2 ... k+1 1 k+2 2 ... n-2 n-k-2 n-1 n-k-1 n n-k n-k+1 ... n-3 n-2 n-1 n] and

iii) [1 2 ... k+1 1 k+2 2 ... n-2 n-k-2 n n-1 n-k-1 n-k n-k+1 ... n-3 n n-2 n-1].

The desired canonical sequence is ii), to which i) is equivalent, which is seen by rotating the first element of i) to the end. In iii), n and n-2 do not overlap yet n-2 is in adj(n-1) having degree k+1. Hence for iii) to derive P_n^k, all nodes of adj(n) must be of degree k+1. But this is only possible if n-1= k+2. Thus iii) is equal to

[1 2 ... k+1 1 k+3 k+2 2 3 4 ... k k+3 k+1 k+2],

which by k+2 rotations and a reflection is equivalent to

[1 k+1 ... 2 1 k+2 k+1 k+3 k ... 4 3 2 k+2 k+3],

which by renumbering 2,3,...,k+1 as k+1,k,...,2, leaving 1, k+2 and k+3 alone, is equivalent to

[1 2 ... k+1 1 k+2 2 k+3 3 ... k-1 k k+1 k+2 k+3],

which is the desired canonical sequence. ∎

Note: If k=1, then case iii) for n>k+3 in the above proof would not need n-1= k+2 since n-2 would be the only adjacency of n-1. The extra cases that we have seen before (see Lemma 2.2.6) would result.

As an example, construct P_7^2 beginning with P_3^2, which equals K_3. Adding 4 to [1 2 3 1 2 3] yields [4 1 2 3 1 4 2 3], [1 2 3 1 4 2 3 4] and [1 4 2 3 4 1 2 3] which are all equivalent to [1 2 3 1 4 2 3 4]. Adding 5 results in [5 1 2 3

1 4 2 5 3 4], [1 2 3 1 4 2 5 3 4 5] and [1 2 3 1 5 4 2 5 3 4]. The first two are clearly equivalent and the last is equivalent to [1 3 2 1 4 3 5 2 4 5] which can be renumbered as [1 2 3 1 4 2 5 3 4 5]. Adding 6 results in [6 1 2 3 1 4 2 5 3 6 4 5], [1 2 3 1 4 2 5 3 6 4 5 6] and [1 2 3 1 4 2 6 5 3 6 4 5]. Again, the first two are clearly equivalent. The last has 3 and 6 overlapping which does not yield P_6^2, since the maximum degree criterion was violated, and thus is excluded. Finally, adding 7 results in the unique canonical sequence of [1 2 3 1 4 2 5 3 6 4 7 5 6 7] that derives P_7^2.

P_n^k, for $2<k<n-1$, is uniquely represented by a circle with chords where the chords go 'around' the circle (see Figure 2.3.1).

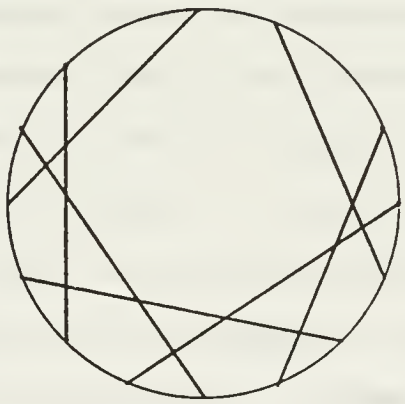


Figure 2.3.1 [1 2 3 1 4 2 5 3 6 4 7 5 6 7] derives P_7^2.

COROLLARY 2.3.2 P_7^2+x is not a circle graph.

This latter result can also be easily seen in that a circle graph containing a focus--a node whose neighborhood is the entire graph--is a permutation graph, but P_7^2+x has no transitive orientation. That is, the unordered pairs that form the edges of P_7^2 cannot be ordered such that if (a,b) and (b,c) are ordered edges, then so is (a,c).

LEMMA 2.3.3 [1 k+3 2 k+4 ... k 2k+2 k+1 1 k+2 2 ... 2k+2 k+2] is equivalent to the unique canonical sequence that derives C_{2k+2}^k, for k>2.

Proof: C_{2k+2}^k is a graph where every node is connected to all other nodes but one. Clearly the sequence derives C_{2k+2}^k, for k>2.

Consider a circle graph sequence S that derives C_{2k+2}^k, for k>2. For every interval i there is a unique interval j that does not overlap i. Thus either S is [... i ... i ... j ... j ...] or [... i ... j ... j ... i ...], where ... represents undetermined elements. Yet i and j must overlap all other intervals, so S is either [i ... i j ... j] or [... i j ... j i ...], where ... represents, in both instances of the first possibility and in the middle instance of the second, 2k elements.

Choose a canonical sequence S that derives C^k_{2k+2}. Renumber the interval that does not overlap 1 as k+2. Rotate the result so that 1 is the first element and no endpoint of k+2 is inside 1. By the above discussion S is equivalent to [1 ... 1 k+2 ... k+2], where ... represents 2k elements. Renumber this such that the second element is k+3 and the interval that does not overlap it is renumbered as 2. Again, by the above discussion, S is equivalent to [1 k+3 2 ... 1 k+2 2 k+3 ... k+2]. Iteratively continue this with k+i+1 and i for i= 3,4,...,k+1. Thus S is equivalent to [1 k+3 2 k+4 ... k 2k+2 k+1 1 k+2 2 ... 2k+2 k+2], and therefore S is the unique canonical sequence that derives C^k_{2k+2}. ∎

THEOREM 2.3.4 [1 n-k+1 2 n-k+2 ... i n-k+i i+1 i+2 ... k k+1 1 k+2 2 ... n-k+i n-2k+i n-2k+i+1 n-2k+i+2 ... n-k] is equivalent to the unique canonical sequence that derives $C^k_n - \{n,n-1,\ldots,n-k+i+1\}$, for $k>2$, $n>2k+3$ and i= 0,1,...,k. (Note: $C^k_n - \{n,n-1,\ldots,n-k+i+1\}$ is the graph of n-k+i nodes labeled 1,2,...,n-k+i and edges (i,j) iff $\|i-j\|<k$, modulo n.)

Proof: Clearly the sequence derives $C^k_n - \{n,n-1,\ldots,n-k+i+1\}$, for $k>2$, $n>2k+3$ and i= 0,1,...,k.

Let $k>2$ and $n>2k+3$ be given. We shall proceed by induction on i. The initial case of i= 0 is true by Lemma 2.3.1 since $C^k_n - \{n,n-1,\ldots,n-k+1\} = P^k_{n-k}$ and $2<k<n-k-1$.

Assume $i>0$. $C_n^k - \{n,n-1,\ldots,n-k+i+1\} = C_n^k - \{n,n-1,\ldots,n-k+i\}$ plus one node $n-k+i$, where $adj(n-k+i)$= all nodes of minimum degree of $N(y) \cup N(z)$ - one node of maximum degree from $N(z)$, where $y=1$ and $z=n-k+i-1$, or $y=n-k+i-1$ and $z=1$. Choose a canonical sequence S that derives $C_n^k - \{n,n-1,\ldots,n-k+i+1\}$. By induction, removing the interval corresponding to the node $n-k+i$ from S yields a sequence equivalent to

[1 n-k+1 2 n-k+2 ... i-1 n-k+i-1 i i+1 ... k k+1 1 k+2 2 ... k+i i k+i+1 i+1 ... n-k+i-1 n-2k+i-1 n-2k+i n-2i+i+1 ... n-k].

Since $i \geq 1$ and $n \geq 2k+3$, we have $k+1 < k+i < n-k+i-3$. Thus the subsequence [k+i i k+i+1 i+1] is correctly placed, and i+1 and n-k+i-1 do not overlap.

Since renumbering 1,2,...,n-k+i-1 as n-k+i-1,n-k+i-2,...,1 and reflecting results in the same sequence, WLOG consider adding interval n-k+i such that n-k+i overlaps all intervals corresponding to the nodes of minimum degree of N(1) U N(n-k+i-1) - one node of maximum degree from N(n-k+i-1). In particular, n-k+i must overlap i and n-k+i-1 and not i+1, and yet overlap a total of k+i intervals. There is only one way to do this:

[1 n-k+1 2 n-k+2 ... i-1 n-k+i-1 i <u>n-k+i</u> i+1 ... k k+1 1 k+2 2 ... k+i i k+i+1 i+1 ... n-k+i-1 n-2k+i-1 <u>n-k+i</u> n-2k+i n-2k+i+1 ... n-k].

Before the left endpoint of n-k+i there are 2i-1 distinct endpoints and after the right endpoint an additional k-i+1, making a total of k+i. Thus S is equivalent to

[1 n-k+1 2 n-k+2 ... i n-k+i i+1 i+2 ... k k+1 1 k+2 2 ... n-k+i n-2k+i n-2k+i+1 n-2k+i+2 ... n-k],

and therefore S is the unique canonical sequence that derives $C_n^k - \{n,n-1,\ldots,n-k+i+1\}$. ∎

It is informative to see why the n= 2k+2 case fails for Theorem 2.3.4. Consider $C_8^3 - \{8,7\}$ which is P_5^3 plus a node 6 where adj(6)= {1,3,4,5} or {1,2,4,5}. (Note: adj(6)= {1,2,3,5} is a symmetric case to {1,3,4,5}.) However the circle graph sequence that derives P_5^3 is [1 2 3 4 1 5 2 3 4 5], which has i+1 (2) and n-k+i-1 (5) overlapping, and n (6) is not restricted from overlapping i+1. Many possibilities occur when adding interval 6, two of which are:

i) [1 2 6 3 4 1 5 6 2 3 4 5] and

ii) [1 2 6 3 4 1 5 2 3 6 4 5],

which are respectively equivalent to the canonical sequences

i) [1 2 3 4 5 1 6 2 3 5 4 6] and

ii) [1 2 3 4 5 1 6 2 4 3 5 6].

Hence there are at least two canonical sequences that derive $C_8^3 - \{8,7\}$.

COROLLARY 2.3.5 [1 n-k+1 2 n-k+2 ... k n k+1 1 k+2 2 ... n n-k] is equivalent to the unique canonical sequence that derives C_n^k, for k>1 and n>2k+1. (Note: for n≤2k+1, $C_n^k = K_n$.)

Proof: Case k=1 follows from Theorem 2.2.8, k>2 and n=2k+1 from Lemma 2.2.4, k>2 and n=2k+2 from Lemma 2.3.3, and k>2 and n>2k+3 from Theorem 2.3.4 with i=k. ∎

C_n^k is uniquely represented by a circle with chords where the chords go 'around' the circle (see Figure 2.3.2).

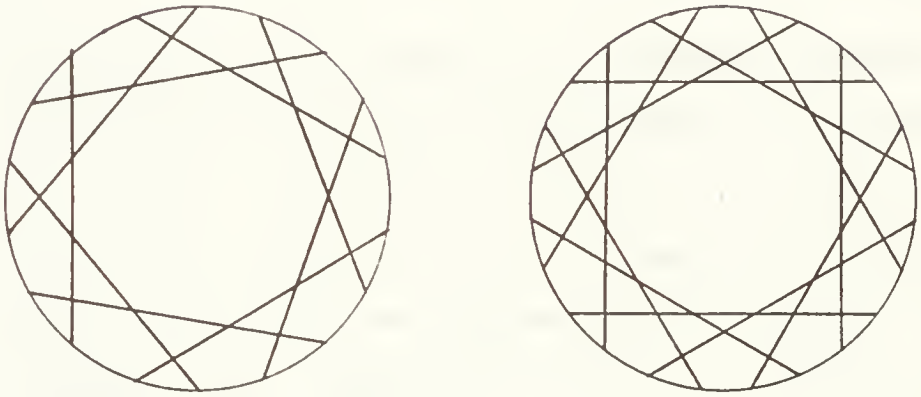


Figure 2.3.2 C_9^2 and C_{12}^3 as

[1 8 2 9 3 1 4 2 5 3 6 4 7 5 8 6 9 7] and

[1 10 2 11 3 12 4 1 5 2 6 3 7 4 8 5 9 6 10 7 11 8 12 9].

2.4 COUNTING PATHS

Some graphs, such as the five sided wheel, have no circle with chords representation. Other graphs, such as the chordless cycle, have a unique circle with chords representation. But other graphs, such as the chordless path of length 5, have more than one circle with chords representation. In this section we shall count the number of

ways a chordless path of length n can be represented on a circle with chords.

LEMMA 2.4.1 P_n, for $n>5$, has $2^{n-5} + 2^{ceil(n/2)-3}$ different circle with chords representations, up to rotations and reflections.

Proof: Let's begin to count the number of ways that such a circle with chords could be constructed. Every circle with chords that derives P_n can be shifted and rotated around so that the two chords corresponding to the endpoints are parallel, one at the top of the circle and the other on the bottom. Let's refer to these as the top and bottom chords, respectively, and the four arcs of the circle defined by these two chords as the top, left, bottom and right sides.

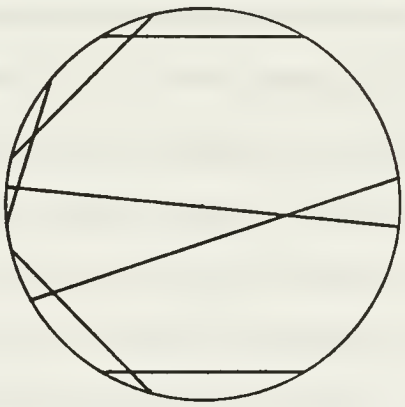


Figure 2.4.1 A shifted and rotated circle with chords that derives P_7.

The chord intersecting the top chord begins at the top side and ends at either the left or right side, but these are equivalent under reflection.

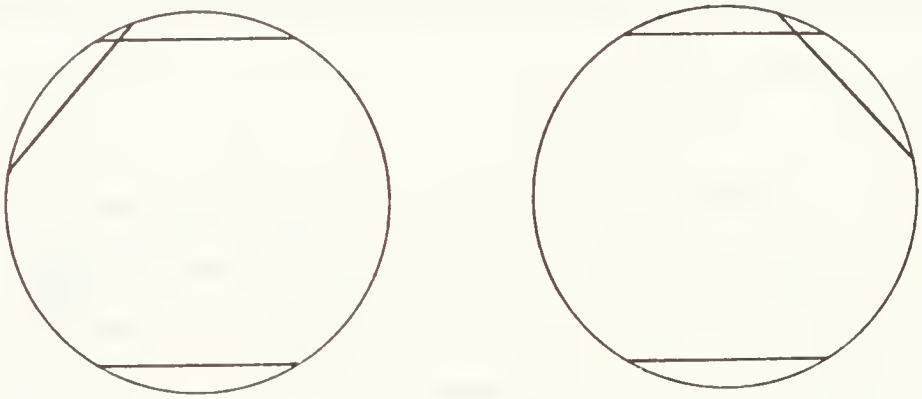


Figure 2.4.2 The possibilities for the top and bottom chords, and the chord intersecting the top.

Let's always choose the top to left side orientation. Once that is fixed, the chord intersecting the bottom chord is also fixed since it must connect the bottom chord to the rest of the chords and there will always be only one way to do that. Hence it will be only those chords between the two end pairs that give us the combinatorial count. Such a chord begins and ends on either the left or right side, or it goes across from left to right or right to left as the path goes from top to bottom. Label these chords as <u>side</u> and <u>across chords</u>, respectively.

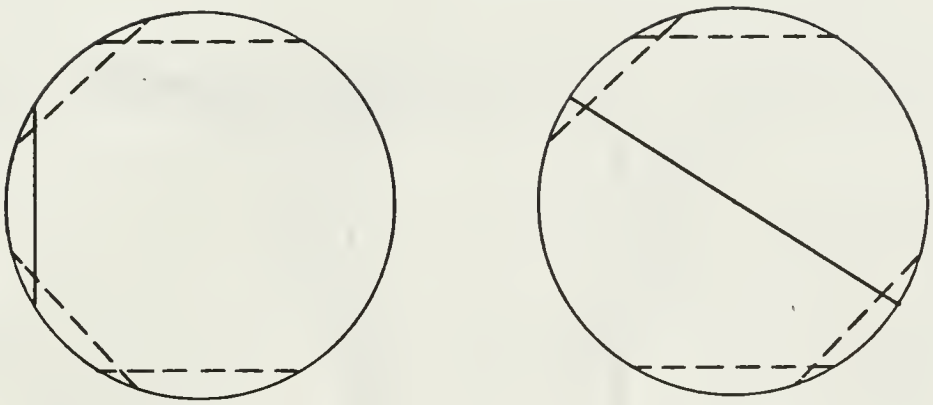


Figure 2.4.3 A side chord and an across chord.

Every circle with chords that derives a chordless path can be represented by a sequence of s's and a's, standing for a side chord and an across chord, respectively, and x's, meaning no orientation is applicable. For example, the two P_5's of Figure 2.4.3 are represented by xxsxx and xxaxx, respectively.

The problem of counting the number of different circles with chords is reduced to counting the number of different sequences of xx...xx, the ... being an arbitrary sequence of s's and a's, where the reversal of xx...xx is not counted a second time: That is, the number, A, of xx...xx where the reversal is the same plus 1/2 the number, B, of xx...xx where the reversal is different. Since

$$A = 2^{\text{ceil}((n-4)/2)}, \text{ and thus } B = 2^{n-4} - 2^{\text{ceil}((n-4)/2)},$$

the total

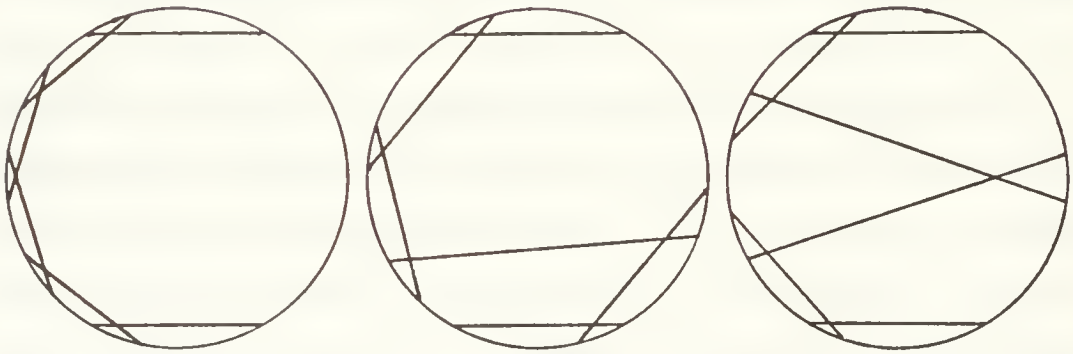
$$A + 1/2 * B = 2^{\text{ceil}((n-4)/2)} + 1/2\ (2^{n-4} - 2^{\text{ceil}((n-4)/2)})$$

$$= 2^{n-5} + 1/2 * 2^{\text{ceil}((n-4)/2)}$$

$= 2^{n-5} + 2^{ceil(n/2)-3}$. ∎

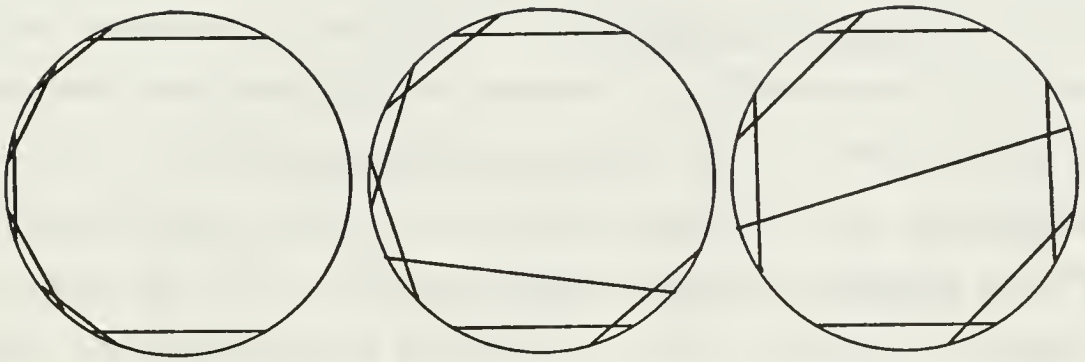
n	# of circles with chords
1	1
2	1
3	1
4	1
5	2
6	3
7	6
8	10
9	20
10	36
11	72
12	136
13	272
14	528
15	1056

Figure 2.4.4 The number of circles with chords deriving P_n.

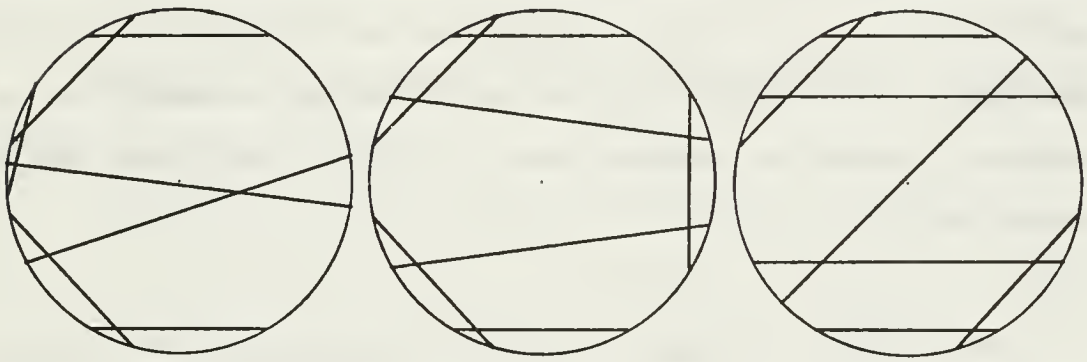


xxssxx xxsaxx xxaaxx

Figure 2.4.5 The three circles with chords deriving P_6.



xxssssxx xxssaxx xxsasxx



xxsaaxx xxasaxx xxaaaxx

Figure 2.4.6 The six circles with chords deriving P_7.

2.5 CONNECTIVITY

Given a graph or a circle with chords it is easy to see whether or not the graph or the derived graph is connected. However, given a circle graph sequence the connectivity is not so obvious. Before we characterize connectivity with respect

to circle graph sequences, we shall need some additional notions of subsequences. Let S and S' be arbitrary sequences. S' is a contiguous subsequence of S if S' is a subsequence of S and all the elements of S between the first and last elements of S' are in S'. S' is a complete subsequence of S if S' is a subsequence of S and every element of S' occurs exactly twice. S' is a complete contiguous subsequence of S if S' is both a complete subsequence and a contiguous subsequence of S. For example, [1 2 1 3 3 2 4 5 4 5] has [3 3], [1 2 1 3 3 2], [4 5 4 5], and [1 2 1 3 3 2 4 5 4 5] as the only complete contiguous subsequences. [1 2 1 2] is a complete subsequence but is not contiguous. [1 2] is a contiguous subsequence but is not complete. It is clear that a circle graph sequence is a complete contiguous subsequence of itself, although not a proper one.

LEMMA 2.5.1 A complete contiguous subsequence of a circle graph sequence S derives a union of connected components of the derived graph of S.

Proof: Let S' be a complete contiguous subsequence of S. Thus any interval of S not in S' overlaps no interval of S'. Therefore S' derives a union of connected components. ∎

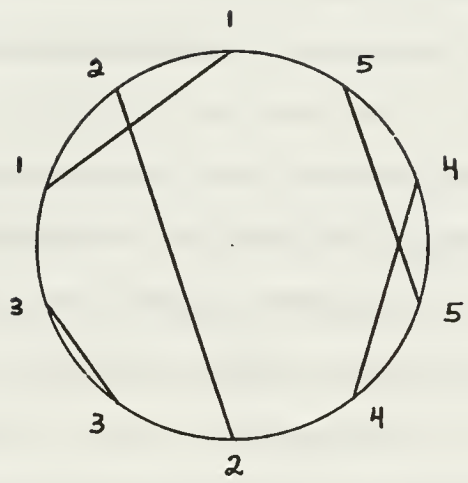


Figure 2.5.1 The proper complete contiguous subsequences of [1 2 1 3 3 2 4 5 4 5] are [3 3], [1 2 1 3 3 2] and [4 5 4 5].

LEMMA 2.5.2 Let S be a circle graph sequence. Then the following are equivalent:

i) S contains a proper complete contiguous subsequence,

ii) it is possible to add an additional interval w to S such that 1) w overlaps no interval of S and 2) the endpoints of w are not adjacent nor at the opposite ends of S, and

iii) S derives a disconnected graph.

Proof: i) → ii) Add interval w to S such that it contains exactly the proper complete contiguous subsequence.

ii) → iii) Observe S+w. Let A be the set of intervals of S contained in w, and let B be the complement. Clearly A and B are nonempty. And no endpoint of any interval of B occurs in any interval of A since w does not overlap any interval in S.

Therefore the nodes corresponding to the intervals of A and those of B are not connected in the derived graph of S.

iii) → i) Pick the interval of S having the leftmost endpoint and mark it, and iteratively mark all intervals overlapped by a marked interval. Clearly the marked intervals correspond to a connected component of the derived graph of S. Since the derived graph is disconnected there are unmarked intervals remaining. Choose S', a maximal contiguous collection of endpoints of unmarked intervals. We claim that S' is a proper complete contiguous subsequence of S.

Clearly S' is a proper contiguous subsequence of S. Assume S' is not complete, that is, there is an interval i having exactly one endpoint in S'. Since S' is maximal, the interval i must contain some endpoints of marked intervals; but not of all marked intervals, for example, the first marked interval. Yet since interval i is unmarked, it must contain either both or no endpoints of a marked interval. Let A be the set of marked intervals contained in interval i, and let B be the complement of the marked intervals. Thus A and B are nonempty, and since interval i is unmarked no interval of A overlaps an interval of B. However, the intervals of A and B correspond to a connected component. →← ∎

COROLLARY 2.5.3 Let C be the intervals corresponding to a connected subgraph of the derived graph of a circle graph sequence S. Then any additional interval w that is added to S, such that w overlaps no interval of C, either contains all or none of the endpoints of the intervals of C. Similarly, any interval v in S that overlaps no interval of C either contains all or none of the endpoints of the intervals of C.

Proof: Let S' be the subsequence of S whose intervals comprise C. Thus S' derives a connected graph. By statement ii) of Lemma 2.5.2, w contains all or none of the endpoints of the intervals of C. ∎

COROLLARY 2.5.4 Let C be the intervals corresponding to a connected component of the derived graph of a circle graph sequence S. Then any additional interval w that is added to S, such that w minimally contains the intervals of C, overlaps no interval of S. Similarly, any interval v in S that minimally contains the intervals of C overlaps no interval of S.

Proof: Observe S+w. Clearly no interval of C overlaps w. By Corollary 2.5.3, all other intervals not in C must contain all or none of the endpoints of the intervals of C. If such an interval contains all of C then it also contains w. If such an interval contains none of C then it is itself contained in w or is disjoint from w. In all cases, there is no interval of S that overlaps w. ∎

2.6 SEPARABILITY

We now return to the notion of separability introduced prior to Lemma 2.2.7. Given a collection of chords, two chords are separated if they cannot both be intersected by a third chord without the third intersecting a fourth chord. Determining whether two chords are separated reduces to the question: Can a chord be added that intersects exactly these two chords? Similarly, recall that two intervals are separated if they cannot both be overlapped by a third interval without the third overlapping a fourth interval. In the two circles with chords that derive P_5, Figure 2.2.4, one has the two end chords separated, and the other does not. Another interesting example is the graph K_4, whose unique canonical sequence is [1 2 3 4 1 2 3 4]. Intervals 1 and 3, and also 2 and 4, are separated. One more definition: Two intervals are in the same connected component of a circle graph sequence S if their corresponding nodes are in the same connected component in the derived graph of S.

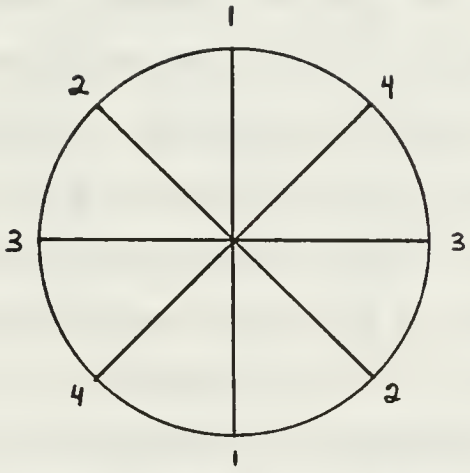


Figure 2.6.1 Chords 1 and 3 are separated since any additional chord that would intersect 1 and 3 would also intersect 2 or 4.

LEMMA 2.6.1 Let S be a circle graph sequence containing [x x y y] as a subsequence. Construct S' by removing [x x y y] and replacing it, position for position, by [v w w v]. Then x and y are separated in S iff v and w are in the same connected component of S'.

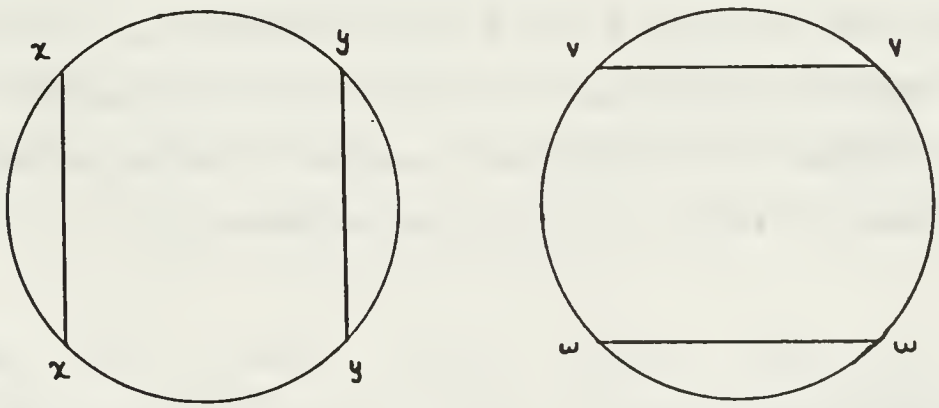


Figure 2.6.2 [x x y y] in S and [v w w v] in S'.

Proof: (→) Assume v and w are in the same connected component of S'. Let C be the intervals corresponding to this connected component. Add an additional interval z to S' such that [v z w w z v] is a subsequence of S'+z. By Corollary 2.5.3, since z does not contain all or none of the endpoints of the intervals of C, z must overlap some interval of C. But z does not overlap v or w, hence z must overlap an interval that was originally in S. For the subsequence [x x y y] in S this implies that the adding of any additional interval z, such that z overlaps both x and y, results in z overlapping some other interval of S. Therefore x and y are separated in S.

(←) Assume v and w are not in the same connected component in S'. Let C be the intervals corresponding to the connected component that contains w. By Corollary 2.5.3, since w is contained in v, all of the endpoints of the intervals of C are contained in v. Add the additional interval z to S' such that z minimally contains C, thus [v z w w z v] is a subsequence of S'+z. By Corollary 2.5.4, z does not overlap any interval of S'. For the subsequence [x x y y] in S this implies that this same adding of the additional interval z overlaps no interval but x and y. Therefore x and y are not separated. ∎

In more intuitive terms, Lemma 2.6.1 says: Non-intersecting chords x and y are separated iff the arcs between them are connected by a sequence of intersecting chords.

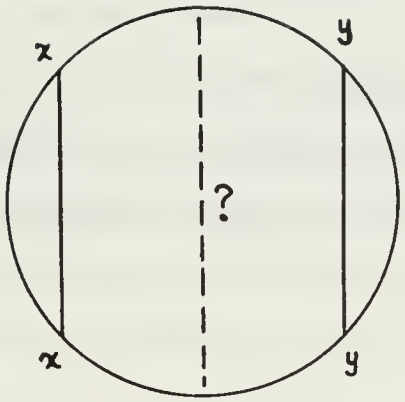


Figure 2.6.3 Non-intersecting chords x and y are separated iff the arcs are connected.

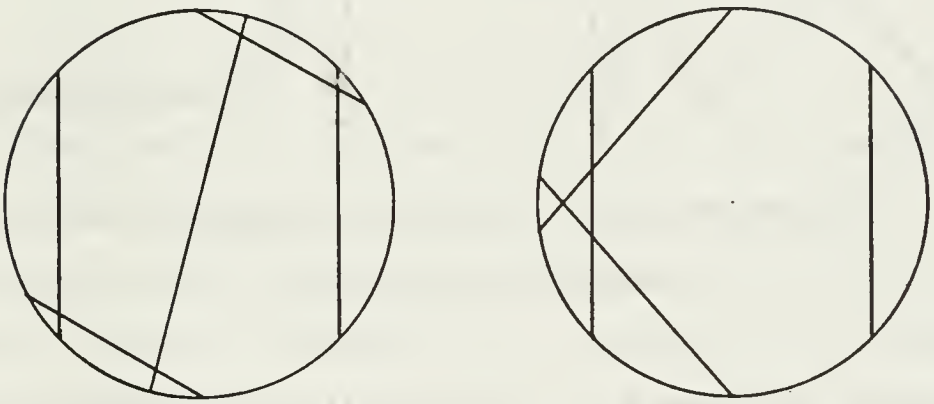


Figure 2.6.4 Two examples of separated chords: i) the two end chords of P_5 where the two arcs between them are connected by a sequence of one chord; and ii) two parallel chords where the two arcs between them are connected by a sequence of two intersecting chords.

LEMMA 2.6.2 Let S be a circle graph sequence containing [x y x y] as a subsequence. Construct S' and S'' by removing [x y x y] and replacing it, position for position, by [v v w w]

and [v w w v], respectively. Then the following are equivalent:

i) x and y are separated in S,

ii) v and w are separated in S' and S'', and

iii) v and w are in the same connected component of S' and S''.

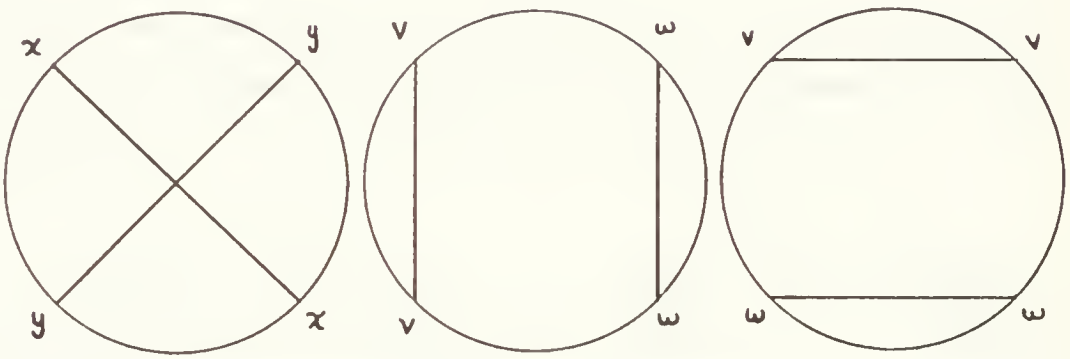


Figure 2.6.5 [x y x y] in S, [v v w w] in S' and [v w w v] in S''.

Proof: By Lemma 2.6.1, ii) and iii) are equivalent.

i) → ii) WLOG assume v and w are not separated in S', and observe the subsequence [v v w w] in S'. Thus it is possible to add an additional interval z such that z overlaps only v and w in S'. For the subsequence [x y x y] in S this implies that this same adding of the additional interval z overlaps only x and y. Therefore x and y are not separated.

ii) → i) Add an additional interval z to S such that z overlaps x and y, and observe the subsequence [x y x y] in S.

Case 1) Interval z contains the two left endpoints, or symmetrically the two right endpoints, of intervals x and y. For the subsequence [v w w v] in S'' this same adding of the additional interval z causes z to overlap some other interval u in S'', since v and w are separated in S''. Thus in S, z would overlap u.

Case 2) Interval z contains the left endpoint of y and the right endpoint of x. For the subsequence [v v w w] in S' we have the same situation as S'' in case 1).

Therefore x and y are separated in S. ∎

Again, in more intuitive terms, Lemma 2.6.2 says: Intersecting chords x and y are separated iff the opposite arcs between them are connected by a sequence of intersecting chords.

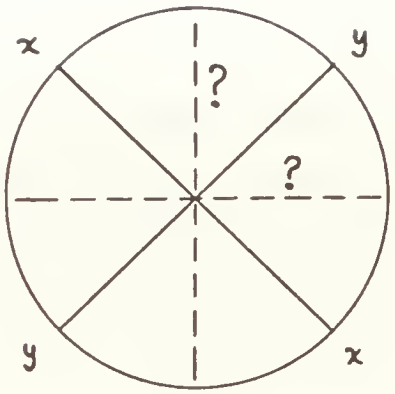


Figure 2.6.6 Intersecting chords x and y are separated iff the opposite arcs are connected.

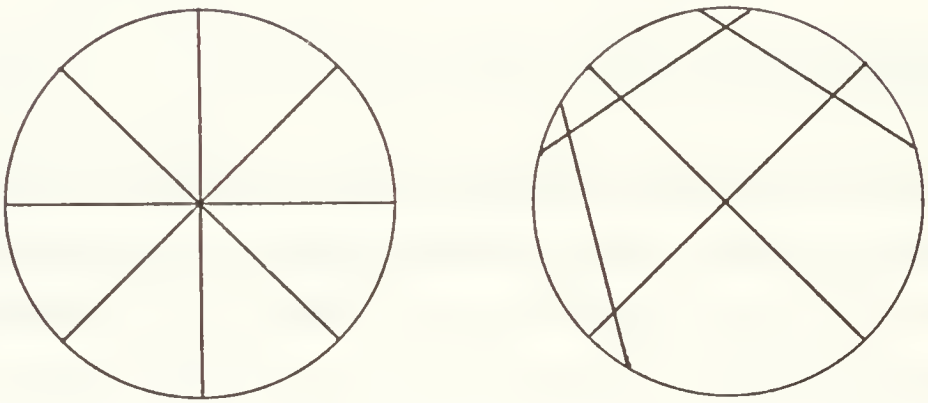


Figure 2.6.7 Two examples of separated chords: i) the two chords of K_4 forming the '×' where the top and bottom arcs, and the left and right arcs, are each connected by a sequence of one chord; and ii) two chords forming the '×' where the top and bottom arcs, and the left and right arcs, are each connected by a sequence of two intersecting chords.

LEMMA 2.6.3 Let S be a circle graph sequence. If intervals x and y are not separated in S and y overlaps only interval z, then x and z are not separated in S.

Proof: Since y and z overlap and not x and y, S contains a subsequence equivalent to [x x y z y] where x, y and z are not renumbered. Since x and y are not separated, it is possible to add an additional interval w such that w overlaps only x and y. Thus S+w contains a subsequence equivalent to [x w x y w z y] with x, y and z the same as before. Since y and w overlap no other interval, all endpoints between the right endpoints of w and y are paired except z. Thus moving the right endpoint of w so that it immediately follows the right endpoint of y results in w overlapping only x and z. Therefore x and z are not separated in S. ∎

THEORM 2.6.4 Let S be a circle graph sequence. If intervals x and y are not separated in S and y overlaps exactly z, or x and z, then x and z are not separated in S.

Proof: The case of y overlapping exactly z follows from Lemma 2.6.3. The case for y overlapping exactly x and z can be seen by adding an interval w whose endpoints are adjacent to those of y without w overlapping y. ∎

2.7 RECOGNITION

We have seen that not all graphs are circle graphs; for example, C_5+x (Corollary 2.2.9) and P_7^2+x (Corollary 2.3.2). Other graphs are known not to be circle graphs, but before we discuss them let us turn our attention to some additional properties of circle graphs.

LEMMA 2.7.1 The neighborhood of every node of a circle graph is a permutation graph.

Proof: Observe any circle with chords representation of a neighborhood N(x) of a circle graph. The chord corresponding to x divides the endpoints of the other chords into two sides with an endpoint of each chord on each side. This produces a matching diagram whose derived intersection graph is a permutation graph. ∎

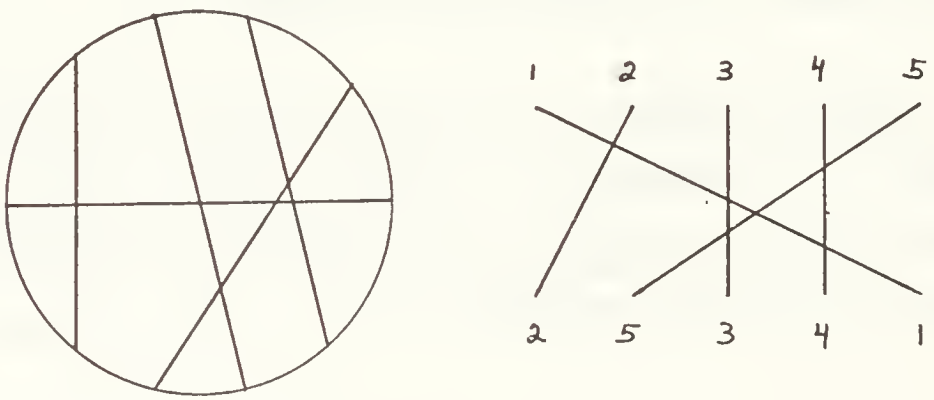


Figure 2.7.1 A neighborhood in a circle with chords and a corresponding matching diagram and permutation.

The converse of Lemma 2.7.1 is not true. Consider the graph in Figure 2.7.2.

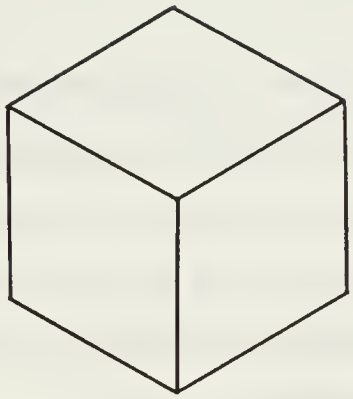


Figure 2.7.2 A graph of 7 nodes where every neighborhood is a permutation graph yet the graph is not a circle graph.

The graph of Figure 2.7.2 also has the property that every proper subgraph is a circle graph.

LEMMA 2.7.2 A graph is a circle graph iff the biconnected components are circle graphs.

Proof: (→) Clearly any subgraph of a circle graph is a circle graph, including each biconnected component.

(←) Let G be a graph whose biconnected components are circle graphs. We shall proceed by induction on the number n of biconnected components. If n=1, it is true. Assume n>1.

Choose B, a biconnected component of G such that G-B intersects B by at most one point. By induction G-B is a circle graph. Construct C, a circle with chords that derives G-B.

Case i) B intersects G-B. Locate the chord x that corresponds to this intersection. Slide all endpoints of C, except x, to one end of x such that the amount of arc of the circle taken is infinitesimal. Now construct B utilizing x. Clearly this can be done without consideration to the rest of C. The resulting circle with chords derives G.

Case ii) B does not intersect G-B. Do the same as in case i) except that there is no chord x and that any point of the arc will do as a point to which to slide all endpoints. ∎

Henceforth, we shall consider only biconnected graphs.

Since every biconnected graph of size 3 or more contains a chordless cycle, which by Theorem 2.2.8 is uniquely constructable as a circle with chords, one way to obtain a circle with chords representation of a graph is to begin with such a cycle and to add one chord at a time until finished, or until an impossibility arises. Along the way many choices might be made as to where to place each chord. Hence the encountering of an impossibility leads to backtracking. This method is at best exponential. There is no known polynomial algorithm for recognizing circle graphs. However many chords have only one possible position for a chord, and in such a

procedure, similar to the one above, we will want to capitalize on this.

LEMMA 2.7.3 Let C be a circle with chords that derives a chordless cycle. Then the number of ways to add a chord such that it intersects chords in C is indicated by the following chart:

# chords	C_n:n>4	C_4	C_3
1	2 sym.	2 sym.	2 sym.
2	1	1	2 sym.
3	2 sym.	2 sym.	3 sym.
4	1	2 sym.	-
1+1	1*	3 sym.	-
2+1	1	-	-
2+2	1	-	-
3+1	0	-	-
1+1+1	0	-	-
2+1+1	0	-	-
all others	0	-	-

* if the two chords are only one apart: 2 sym.

where singleton digits refer to the number of consecutive nodes of the cycle, and the "+" indicates a partition into disconnected sets of consecutive nodes.

Proof: This follows by observing the circles with chords that derive chordless cycles, which by Theorem 2.2.8 are uniquely representable. ∎

Lemma 2.7.3 gives us a whole collection of forbidden circle graph subgraphs. The more notable are shown in Figure 2.7.3.

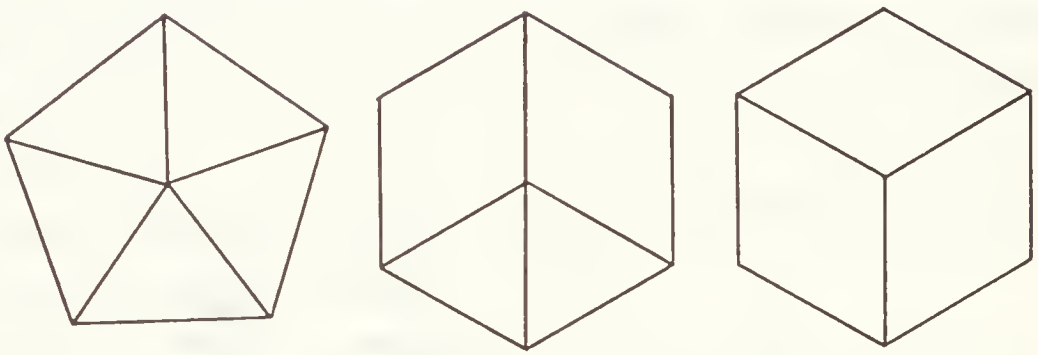


Figure 2.7.3 Three forbidden circle graph subgraphs: C_5 with a 5, C_6 with a 3+1, and C_6 with a 1+1+1.

Lemma 2.7.3 by no means exhausts all possible forbidden subgraphs. Consider S_3, the stable set of three nodes. There are only two circle graph sequences that derive S_3: [1 1 2 2 3 3] and [1 2 3 3 2 1]. However, if a single interval is to overlap all three intervals, such as in $K_{1,3}$, then [1 2 3 3 2 1] is the only subsequence for the S_3 subgraph of $K_{1,3}$. On the other hand [1 1 2 2 3 3] may be required by other constraints. Three independent nodes form an <u>asteroidal triple</u> (Lekkerkerker and Boland [1962]) if there are three paths each connecting a different pair of the nodes without passing through the neighborhood of the third node. Clearly [1 2 3 3 2 1] cannot derive an asteroidal triple since any path from 1 to 3 passes through the neighborhood of 2. Therefore an asteroidal triple

is uniquely represented by [1 1 2 2 3 3]. A node connected to an asteroidal triple forms an asteroidal star. We have the following result:

LEMMA 2.7.4 The asteroidal star is a forbidden circle graph subgraph.

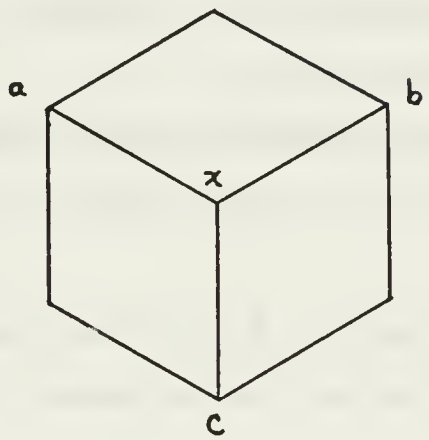


Figure 2.7.4 A forbidden circle graph subgraph: x, a, b and c form an asteroidal star.

THEOREM 2.7.5 (Kabell [1980]) Let G be a connected graph. Form the graph H by making two copies of G and connecting the pairs of the corresponding nodes between the copies (i.e., H= G x K_2). Then H is a circle graph iff G is a chordless path.

Proof: We offer the following proof.

(←) If G is a chordless path, then H appears as a ladder. Construct the first rung of H: [1 2 1 2]. Adding on the second rung can be done by [3 1 4 2 1 3 2 4]. Iteratively

continue adding two endpoints about the first and last endpoint: i.e., the next step would be [5 3 6 1 4 2 1 3 2 5 4 6]. The result derives H.

(→) If G is not a chordless path, then either G contains a $K_{1,3}$ subgraph or a cycle.

Case i) G contains a $K_{1,3}$ subgraph. But then H contains $K_{1,3}$ x K_2, which in turn contains an asteroidal star. By Lemma 2.7.4, H is not a circle graph.

Case ii.1) G contains a chordless cycle of size > 4. But then H contains C_n x K_2, for n>4, which again in turn contains an asteroidal star.

Case ii.2) G contains K_3 x K_2, which is the complement of C_6, which by observation is not a circle graph. ∎

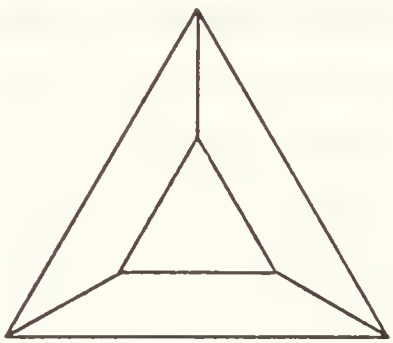


Figure 2.7.5 $\overline{C}_6$ is not a circle graph.

Lemmas 2.7.3, 2.7.4 and Theorem 2.7.5 do not yet give all possible forbidden circle graph subgraphs; for example, P_7^2+x (Corollary 2.3.2).

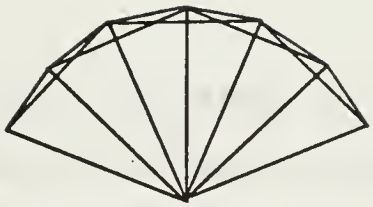


Figure 2.7.6 P_7^2+x, a forbidden circle graph subgraph not mentioned in Lemma 2.7.3, 2.7.4 or Theorem 2.7.5.

There is no known forbidden subgraph characterization of circle graphs.

CHAPTER 3: THE BERGE STRONG PERFECT GRAPH CONJECTURE

In this chapter we shall show that the Berge Strong Perfect Graph Conjecture holds for the class of circle graphs. We shall also give a new, very simple proof that the conjecture holds for $K_{1,3}$-free graphs. Parthasarathy and Ravindra [1976] were the first to show this latter result. But first we will introduce Berge's conjecture and the work that has been done in an attempt to solve it. Many of these results will be used in our proofs of the conjecture for circle graphs and $K_{1,3}$-free graphs.

3.1 p-CRITICAL AND PARTITIONABLE GRAPHS

In the early 1960's, Claude Berge introduced the notion of a perfect graph and put forth a number of conjectures characterizing such a graph.

Given a graph G let $\alpha(G)$ be the size of the maximum stable set, $\omega(G)$ be the size of the maximum clique, $\gamma(G)$ be the size of the smallest stable set cover, i.e., a coloring, and $\theta(G)$ be the size of the smallest clique cover. Clearly the following hold:

i) $\alpha(G) \leq \theta(G)$ and

ii) $\omega(G) \leq \gamma(G)$.

The condition for being perfect is when the equality of i) and the quality of ii) holds for a graph and all of its subgraphs. G is called

α-perfect if $\alpha(H) = \theta(H)$ for all H contained in G and

γ-perfect if $\omega(H) = \gamma(H)$ for all H contained in G.

In 1972, Lászlo Lovász showed that a graph is α-perfect iff it is γ-perfect. He also showed a third equivalent condition:

$$\alpha(H)\omega(H) \geq |H| \text{ for all } H \text{ contained in } G.$$

These results of Lovász have come to be known as the Perfect Graph Theorem. Padberg [1976] derived these same results, and others, using the methods of linear programming.

A graph G is p-critical if it is minimally imperfect, that is, it is imperfect and every proper subgraph is perfect. The only known p-critical graphs are the odd chordless cycles of size ≥ 5 and their complements. Berge has conjectured this to be the only case, a conjecture that has come to be known as the Berge Strong Perfect Graph Conjecture:

A graph is p-critical iff it is an odd chordless cycle of size ≥ 5 or the complement of one.

Another useful form is:

A graph is imperfect iff it contains an odd chordless cycle of size ≥ 5 or the complement of one.

Some properties of p-critical graph are immediate.

LEMMA 3.1.1 If G is p-critical having n nodes, then $n= \alpha(G)\omega(G)+1$ (Lovász [JCT 1972]), and $\alpha(G)= \alpha(G-x)$, $\omega(G)= \omega(G-x)$, $\gamma(G)= \gamma(G-x)+1$ and $\theta(G)= \theta(G-x)+1$ for all nodes x (Berge [1961] and [1962]).

Proof: Choose any node x. By the Perfect Graph Theorem, we have

$$n-1 \leq \alpha(G-x)\omega(G-x) \leq \alpha(G)\omega(G) < n.$$

Hence $n= \alpha(G)\omega(G)+1$. Being p-critical gives us

$$\theta(G-x) = \alpha(G-x) \leq \alpha(G) < \theta(G) \leq \theta(G-x)+1.$$

Thus $\alpha(G)= \alpha(G-x)$ and $\theta(G)= \theta(G-x)+1$. Similarly $\omega(G)= \omega(G-x)$ and $\gamma(G)= \gamma(G-x)+1$. ∎

In 1979, Bland, Huang and Trotter introduced a notion that relates very closely to p-critical graphs. We give a slightly weaker, but equivalent, formulation. Let $\alpha, \omega \geq 2$ be arbitrary integers. A graph G having n nodes is <u>(α,ω)-partitionable</u> if $n= \alpha\omega+1$, and $\alpha \geq \theta(G-x)$ and $\omega \geq \gamma(G-x)$ for all nodes x. (Note: The original formulation used strict equality.) By Lemma 3.1.1, every p-critical graph G is $(\alpha(G),\omega(G))$-partitionable.

LEMMA 3.1.2 (Bland, Huang and Trotter [1979]) If G is (α,ω)-partitionable, then $\alpha= \alpha(G)$ and $\omega= \omega(G)$.

Proof: We offer the following new proof. Since $\gamma(G-x)$ stable sets cover $\alpha\omega$ nodes, some stable set of G is of at least size α. Hence $\alpha \leq \alpha(G)$, and similarly, $\omega \leq \omega(G)$. Choose S, a maximum stable set of G, and y, a node of G not in S. With $\omega(G) \geq 2$, y exists, and S is in G-y. Thus

$$\alpha(G) = |S| = \alpha(G-y) \leq \theta(G-y) < \alpha.$$

Therefore $\alpha = \alpha(G)$, and similarly $\omega = \omega(G)$. ∎

Henceforth, let α denote $\alpha(G)$ and ω denote $\omega(G)$.

THEOREM 3.1.3 The class of p-critical graphs is properly contained in the class of partitionable graphs, which is in turn properly contained in the class of imperfect graphs.

Proof: By Lemma 3.1.1 we know that every p-critical graph is a partitionable graph, and since C_{10}^3 is (3,3)-partitionable and not p-critical because it properly contains C_5, we have the class of p-critical graphs properly contained in the class of partitionable graphs. By Lemma 3.1.2 and the Perfect Graph Theorem, we know that every partitionable graph is imperfect, and since the wheel of five sides is imperfect and not partitionable because $n \neq \alpha\omega+1$, we have the class of partitionable graphs properly contained in the class of imperfect graphs. ∎

THEOREM 3.1.4 (Bland, Huang and Trotter [1979]) G is minimally imperfect (p-critical) iff it is minimally partitionable.

Proof: We offer the following proof.
(→) If G is not minimally partitionable then there is a proper subgraph that is minimally partitionable and hence imperfect. Thus G is not minimally imperfect.
(←) If G is not minimally imperfect then there is a proper subgraph that is minimally imperfect and hence partitionable. Thus G is not minimally partitionable. ∎

We now have a third form of the Berge Strong Perfect Graph Conjecture:

If a graph is partitionable, then it contains an odd chordless cycle of size > 5 or the complement of one.

3.2 PROPERTIES OF PARTITIONABLE GRAPHS

$C^{\omega-1}_{\alpha\omega+1}$ is an (α,ω)-partitionable graph for $\alpha,\omega > 2$ and it contains an odd chordless cycle of size > 5 or the complement of one. If it could be shown that every p-critical graph, or partitionable graph, is a $C^{\omega-1}_{\alpha\omega+1}$ graph then the Berge Strong Perfect Graph Conjecture would be decided in the affirmative. In 1976, Chvátal demonstrated the sufficiency of a weaker condition: If a p-critical graph G has $C^{\omega-1}_{\alpha\omega+1}$ as a spanning

subgraph, for $\alpha = \alpha(G)$ and $\omega = \omega(G)$, then G is an odd chordless cycle of size > 5 or the complement of one. Hence if it could be shown that every p-critical is indeed spanned by $C^{\omega-1}_{\alpha\omega+1}$, our goal would be accomplished.

The only progress made so far in quest of an answer to the Berge Strong Perfect Graph Conjecture is that it has been shown to hold for various classes of graphs. However the proof for each class generally depends heavily on the underlying structure of each class.

planar graphs	Tucker [1973]
circular arc graphs	Tucker [1975]
$K_{1,3}$-free graphs	Parthasarathy and Ravindra [1976]
3-chromatic graphs	Tucker [1977]
toroidal graphs	Grinstead [1978]
(K_4-e)-free graphs	Parthasarathy and Ravindra [1979]

Figure 3.2.1 Some classes of graphs for which the Berge Strong Perfect Graph Conjecture holds.

Since no property has been found that differentiates between a minimally partitionable graph (p-critical) and a non-minimally partitionable graph, efforts to decide the Berge conjecture have been focused on the full class of partitionable graphs. In 1979, Chvátal, Graham, Perold and Whitesides demonstrated construction and design techniques that result in

non-$C^{\omega-1}_{\alpha\omega+1}$ partitionable graphs. Bland, Huang and Trotter, also in 1979, independently demonstrated a non-$C^{\omega-1}_{\alpha\omega+1}$ partitionable graph. Hence the search continues for a property of minimally partitionable graphs that rules out non-$C^{\omega-1}_{\alpha\omega+1}$ partitionable graphs.

Many properties of partitionable graphs have been found and are scattered throughout the literature under the guise of different names, e.g., (α,ω)-graphs, pseudo p-critical graphs. In this section we shall present the more important properties and some new ones, using a unified notation not necessarily that of the original authors. For the sake of completeness, proofs to all of the properties shall be given eventhough the ideas behind some of them have not been changed from the original proofs. An indication will be given where results have been sharpened, a more direct proof given, or where a totally new proof is offered. In any case, all proofs have been rewritten to conform in style. In many cases clarity has been significantly improved.

Let $\underline{\theta(G)}$ denote a minimum clique cover of G and $\underline{\Gamma(G)}$ a minimum stable set cover. Let $\underline{\omega\text{-clique}}$ denote a clique of size ω and $\underline{\alpha\text{-stable set}}$ a stable set of size α.

LEMMA 3.2.1 (Bland, Huang and Trotter [1979]) If G is (α,ω)-partitionable, then, for all nodes x, $\alpha = \theta(G-x)$ and $\omega = \gamma(G-x)$, and every clique of $\theta(G-x)$ is an ω-clique and every

stable set of $\Gamma(G-x)$ is an α-stable set. Furthermore, for every ω-clique C there is an α-stable set S such that $C \cap S = \phi$, and vice versa.

Proof: By Lemma 3.1.2 $\alpha = \alpha(G)$ and $\omega = \omega(G)$. Since $\Theta(G-x)$ covers $\alpha\omega$ nodes and $\theta(G-x) < \alpha$, every clique of $\Theta(G-x)$ must be an ω-clique and $\theta(G-x) = \alpha$. Similarly every stable set of $\Gamma(G-x)$ must be an α-stable set and $\omega = \gamma(G-x)$.

Choose an ω-clique C. Pick a node x of C and observe any $\Gamma(G-x)$. The ω stable sets of $\Gamma(G-x)$ cover the $\omega-1$ nodes of C-x and not x. Hence there is some α-stable set S in $\Gamma(G-x)$ that contains no node of C. Therefore $C \cap S = \phi$. Similarly for each α-stable set S there is an ω-clique C such that $C \cap S = \phi$. ∎

From Lemma 3.2.1 we see that every (α,ω)-partitionable graph can be pictured as an $\alpha \times \omega$ rectangle of nodes plus one extra node, where each row is an ω-clique and each column is an α-stable set.

LEMMA 3.2.2 (Bland, Huang and Trotter [1979]) If G is (α,ω)-partitionable, then, for $n = \alpha\omega+1$,

i) G contains a set of n ω-cliques that covers each node of G ω times,

ii) G contains a set of n α-stable sets that cover each node of G α times, and

iii) each ω-clique intersects all but one α-stable set, and vice versa.

Furthermore, these are the only ω-cliques and α-stable sets of G.

Proof: We offer the following more direct proof. Choose an ω-clique C of G, and for each x in C choose a $\Theta(G-x)$. Construct the $n \times n$ 0-1 matrix A whose first row is the characteristic vector of C and whose subsequent rows are the characteristic vectors of each clique in $\Theta(G-x)$ for each x in C. With ω collections of $\Theta(G-x)$, each having α cliques, we have constructed, together with the original clique, n rows. By Lemma 3.1.2 and 3.2.1, each row sum of A is ω: $A\mathbf{1} = \omega\mathbf{1}$.

Observe each column of A. Every node y not in C is covered by $\Theta(G-x)$ for each x in C, that is, ω times. And every node y in C is covered once by C and by $\Theta(G-x)$ for each $x \neq y$ in C, that is, $1 + (\omega - 1)$ times. Hence, each column sum of A is also ω: $\mathbf{1}A = \omega\mathbf{1}$. Condition i) will be satisfied after we have shown the rows of A to be distinct.

Construct the $n \times n$ 0-1 matrix B where row i is the characteristic vector of an α-stable set that by Lemma 3.2.1 does not intersect the ω-clique represented by row i in A. Hence, $B\mathbf{1} = \alpha\mathbf{1}$ and $AB' \leq J-I$, where J is the $n \times n$ matrix of all ones and I the $n \times n$ identity matrix. However,

$$\mathbf{1}AB' = \omega\mathbf{1}B' = \omega\alpha\mathbf{1} = (n-1)\mathbf{1}.$$

Therefore $AB' = J-I$. Condition iii) is satisfied.

Moreover, since no two rows, or two columns, of $J-I$ are the same, no two rows of A, nor of B, are the same. Hence the rows of A, and the rows of B, are distinct. Condition i) is now satisfied. Also note that since $J-I$ is nonsingular, $(J-I)^{-1} = [1/(n-1)]J-I$, A and B are nonsingular.

With $1B = 1BA'(A')^{-1} = 1(J-I)(A')^{-1} = (n-1)1(A')^{-1} = [(n-1)/\omega]1 = \alpha 1$, condition ii) is satisfied.

For the furthermore, choose any ω-clique and let c be its characteristic vector. We shall show that c is a row of A. Observe $tA = c$.

$$t = cA^{-1} = c[(1/\omega)J-B'] = (1/\omega)cJ-cB' = 1-cB'.$$

Hence t is a 0-1 vector. Yet

$$t1 = (1-cB')1 = n-c(\alpha 1) = n-\alpha\omega = 1.$$

Therefore t is a unit vector and thus c is a row of A. Similarly any α-stable set is a row of B. ∎

The converse of Lemma 3.2.2 was never stated by the authors and we now offer it here.

THEOREM 3.2.3 Let $\alpha, \omega > 2$ and let G be a graph of n nodes. Then G is (α,ω)-partitionable iff the following hold:

i) $n = \alpha\omega+1$,

ii) G contains exactly n ω-cliques and n α-stable sets,

iii) every node of x is contained in exactly ω ω-cliques and α α-stable sets, and

iv) each ω-clique intersects all but one α-stable set, and vice versa.

Proof: ($\rightarrow$) A restatement of Lemma 3.2.2.

($\leftarrow$) We are given that $\alpha, \omega > 2$ and by i) $n = \alpha\omega + 1$. So our goal is to show $\alpha > \theta(G-x)$ and $\omega > \gamma(G-x)$ for all nodes x. Construct the $n \times n$ 0-1 matrices A and B where the rows are characteristic vectors of the n ω-cliques and n α-stable sets, respectively. By iv) there is an arrangement of the rows of B such that $AB' = J-I$. Conditions ii) and iii) imply

$$AJ = JA = \omega J \text{ and } BJ = JB = \alpha J.$$

Observe that $A'B = B^{-1}BA'B = B^{-1}(J-I)B = J-I$.

Choose a node x of G. Let e be the characteristic vector of G-x. Since e is a column of A'B, the same column of B designates α columns of A' whose sum is e. That is, the α ω-cliques corresponding to these α columns cover G-x. Hence $\theta(G-x) < \alpha$. Similarly e is also a row of A'B, and the corresponding row of A' designates ω α-stable sets that cover G-x. Hence $\gamma(G-x) < \omega$. ∎

COROLLARY 3.2.4 (Padberg [1974]) If G is p-critical, then the four conditions of Theorem 3.2.3 hold.

COROLLARY 3.2.5 (Bland, Huang and Trotter [1979]) If G is (α,ω)-partitionable, then $\Theta(G-x)$ and $\Gamma(G-x)$ are unique for each node x. Furthermore, the α α-stable sets that have an empty intersection with some ω-clique of $\Theta(G-x)$ are precisely those α α-stable sets that contain x, and the ω ω-cliques that have an empty intersection with some α-stable set of $\Gamma(G-x)$ are precisely those ω ω-cliques that contain x.

Proof: These results follow from the uniqueness of the n ω-cliques and n α-stable sets and the matrix equation $A'B = J-I$, using our previous notation. The furthermore follows since an α-stable set S contains x iff S intersects exactly $\alpha-1$ ω-cliques of $\Theta(G-x)$. Similarly the result follows for ω-cliques. ∎

From Corollary 3.2.5 we now see that the $\alpha \times \omega$ rectangular representation of an (α,ω)-partitionable graph, where each row is an ω-clique and each column is an α-stable set, is unique up to permutations of the rows and of the columns once the extra node is chosen.

We offer the next new result.

COROLLARY 3.2.6 Let G be (α,ω)-partitionable. Then

i) $\Theta(G-x) \cap \Theta(G-y) \neq \phi$ iff there is an α-stable set containing both x and y, and

ii) $\Gamma(G-x) \cap \Gamma(G-y) \neq \phi$ iff there is an ω-clique containing both x and y.

Proof: ($\rightarrow$) By Corollary 3.2.5 any α-stable set having an empty intersection with some ω-clique of the intersection in i) must contain both x and y. Similarly ii) is true.
($\leftarrow$) Again, by Corollary 3.2.5, any α-stable set containing x and y has an empty intersection with some ω-clique in both $\theta(G-x)$ and $\theta(G-y)$. By Theorem 3.2.3 such an ω-clique is unique and is thus the same ω-clique in $\theta(G-x)$ and $\theta(G-y)$, making the intersection non-empty. Similarly the result follows for any ω-clique in ii). ∎

Theorem 3.2.3 is the best result found so far in the attempt to identify p-critical graphs. Theorem 3.1.4 shows that the minimal part of p-critical graphs needs to be strongly used in order to characterize them. However, a few other properties of partitionable graphs can be deduced from Theorem 3.2.3 and are very useful in showing that the Berge Strong Perfect Graph Conjecture is true for special classes of graphs.

Let C(G) denote the clique graph of G, where every node of C(G) corresponds to a maximum clique in G, and every edge of C(G) corresponds to intersecting maximum cliques.

We sharpen the following result of Tucker.

LEMMA 3.2.7 If G is (α,ω)-partitionable, then C(G) is (α,ω)-partitionable (Tucker [1977]). Furthermore, the clique matrices of G and C(G) are the transpose of one another, and similarly the stable set matrices are the transpose of one another.

Proof: We offer the following new proof. Since $\alpha,\omega \geq 2$ and the number of nodes in C(G) is $\alpha\omega+1$, we need only to show that $\alpha \geq \theta(C(G)-x)$ and $\omega \geq \gamma(C(G)-x)$ for all nodes x of C(G).

Pick a node x of C(G). Choose the corresponding ω-clique C in G and the α-stable set S such that $C \cap S = \phi$. For each node in S there are ω ω-cliques covering it. In C(G) this corresponds to an ω-clique. And since no ω-clique of G covers more than one node of S, the α collections of ω ω-cliques formed by S account for all ω-cliques of G except C. In C(G) this corresponds to α ω-cliques that cover C(G)-x. Hence $\theta(C(G)-x) \leq \alpha$.

For each node z in C, $\theta(G-z)$ corresponds to an α-stable set in C(G). And since, by Corollary 3.2.6, $\theta(G-u) \cap \theta(G-v) = \phi$ for u,v in C, the ω collections of α ω-cliques, formed by $\theta(G-z)$ for each z in C, account for all ω-cliques except C. In C(G) this corresponds to ω α-stable sets that cover C(G)-x. Hence $\gamma(C(G)-x) \leq \omega$. Therefore C(G) is (α,ω)-partitionable.

The furthermore follows by the uniqueness of the clique matrix of C(G) and because the transpose of the clique matrix of G is a valid choice. The result for stable sets follows from the matrix equation: $AB' = J-I$. ∎

Let S(G) denote the skeleton graph of G, that is, the graph obtained from G by removing all edges that are not in a maximum clique of G.

COROLLARY 3.2.8 (Tucker [1977]) If G is (α,ω)-partitionable, then S(G) is (α,ω)-partitionable.

Proof: This follows from the furthermore of Lemma 3.2.7 implying $S(G) = C(C(G))$. We also offer the more direct proof by observing:

i) $\alpha,\omega \geq 2$ and $n = \alpha\omega+1$,

ii) $\alpha \geq \theta(G-x) = \theta(S(G)-x)$, since edges of maximum cliques are not removed, and

iii) $\omega \geq \gamma(G-x) \geq \gamma(S(G)-x)$, since there are fewer edges. ∎

We can always limit the scope of our search for an answer to the Berge Strong Perfect Graph Conjecture to skeletons of graphs--If the conjecture is true the skeleton of a p-critical graph is itself. And if the conjecture is false the skeleton of a counter-example p-critical graph will be neither an odd chordless cycle of size ≥ 5 nor the complement of one, due to Chvátal's [1976] result that a p-critical graph having $C^{\omega-1}_{\alpha\omega+1}$,

for $\alpha, \omega > 2$, as a spanning subgraph is an odd chordless cycle of size > 5 or the complement of one--By Corollary 3.2.8 we know that skeletons of p-critical graphs are partitionable. Hence we need only consider partitionable graphs that have every edge in a maximum clique in our search for an answer to the Berge Strong Perfect Graph Conjecture.

The next few results concern the connectivity of a partitionable graph. Recall that N(x) denotes the neighborhood of a node x.

LEMMA 3.2.9 (Tucker [1977]) If G is (α, ω)-partitionable, then G-N(x) is connected.

Proof: Assume there is a node x such that G-N(x) is disconnected. Consider $\Theta(G-x)$. The removal of N(x) partitions $\Theta(G-x)$ into disconnected components: That is, there exist disjoint sets Θ_1, Θ_2 both non-empty such that $\Theta_1 \cup \Theta_2 = \Theta(G-x)$, and $\bigcup_{C \in \Theta_1} C-N(x)$, $\bigcup_{C \in \Theta_2} C-N(x)$ are not connected. Clearly, $\|\Theta_1\| + \|\Theta_2\| = \|\Theta(G-x)\| = \alpha$.

Choose C_1 in Θ_1 and C_2 in Θ_2, and the α-stable sets S_1, S_2 such that $C_1 \cap S_1 = C_2 \cap S_2 = \phi$. By Corollary 3.2.5 S_1 and S_2 contain x. Hence $S_1 \cap \bigcup_{C \in \Theta_2} C-N(x)$ has size $\|\Theta_2\|$ and $S_2 \cap \bigcup_{C \in \Theta_1} C-N(x)$ has size $\|\Theta_1\|$, and are not connected. Therefore $[S_1 \cap \bigcup_{C \in \Theta_1} C-N(x)] \cup [S_2 \cap \bigcup_{C \in \Theta_2} C-N(x)] \cup \{x\}$ is a stable set of size $\alpha+1$. ++ ∎

LEMMA 3.2.10 (Bland, Huang and Trotter [1979]) If S_1, S_2 are α-stable sets of an (α,ω)-partitionable graph, then $S_1 \Delta S_2$, the symmetric difference, is connected.

Proof: We offer the following new proof. Assume there are α-stable sets S_1, S_2 such that $S_1 \Delta S_2$ is disconnected: That is, there exist disjoint subgraphs G_1, G_2 both non-empty such that $G_1 \cup G_2 = S_1 \Delta S_2$, and G_1, G_2 are not connected. Observe that $\|S_1 \cap G_i\| = \|S_2 \cap G_i\|$, for $i = 1,2$. For example if $\|S_1 \cap G_1\| > \|S_2 \cap G_1\|$ then $(S_1 \cap G_1) \cup (S_2 \cap G_2) \cup (S_1 \cap S_2)$ is a stable set of size $> \alpha$.

Choose the ω-clique C such that $C \cap S_1 = \phi$. Thus $\|C \cap S_2\| = \|C \cap (S_1 \Delta S_2)\| = 1$. Choose i such that $C \cap G_i \neq \phi$. WLOG assume $i = 1$. Then $(S_1 \cap G_1) \cup (S_2 \cap G_2) \cup (S_1 \cap S_2)$ is the same size as $(S_1 \cap G_1) \cup (S_1 \cap G_2) \cup (S_1 \cap S_2)$, and hence is another α-stable set besides S_1 that does not intersect C. ↛↚ ∎

COROLLARY 3.2.11 (Tucker [1977]) If S_1, S_2 are distinct sets of $\Gamma(G-x)$ for any node x of an (α,ω)-partitionable graph G, then $S_1 \cup S_2$ is connected.

A graph is <u>k-connected</u> if the removal of any set of nodes of size $< k$ results in a connected graph. We offer the following new result of (α,ω)-partitionable graphs.

THEOREM 3.2.12 If G is (α,ω)-partitionable, then G is ω-connected.

Proof: Let T be a collection of at most $\omega-1$ nodes of G. Pick any node x from T, and observe G-x. Since $\|T-x\| < \omega-2$, there exist two α-stable sets S_1, S_2 in $\Gamma(G-x)$ such that $(S_1 \cup S_2) \cap T-x = \phi$. By Corollary 3.2.11 $S_1 \cup S_2$ is connected. And since $S_1 \cup S_2$ is in G-T and contains a maximum stable set of G-T, G-T is connected. Therefore G is ω-connected. ∎

The following is another new result of (α,ω)-partitionable graphs that plays a major role in the proofs presented in sections 3.3 and 3.4.

THEOREM 3.2.13 If S_1, S_2 are distinct sets of $\Gamma(G-x)$ for any node x of an (α,ω)-partitionable graph G, then $S_1 \cup S_2 \cup x$ is biconnected.

Proof: Assume there is an S_1, S_2 and x such that $S_1 \cup S_2 \cup x$ is not biconnected. Choose s, an articulation point of $S_1 \cup S_2 \cup x$. Hence there exist disjoint subgraphs G_1, G_2 both non-empty such that $G_1 \cup G_2 = (S_1 \cup S_2 \cup x) - s$, and G_1, G_2 are not connected. By Corollary 3.2.11 $S_1 \cup S_2$ is connected, thus $s \neq x$. WLOG assume s is in S_1 and x is in G_2.

Observe $\theta(G-s)$. Each node of S_1 is paired with a node of S_2, except for one node of S_2 that must be paired with x which is in G_2. Thus $\|S_1 \cap G_1\| = \|S_2 \cap G_1\|$. Observe $\theta(G-b)$ for b in $S_2 \cap G_1$. Each node of $S_1 \cap G_1$ must be paired with a node of $S_2 \cap G_1$, except for one node of $S_1 \cap G_1$ that must be paired with x, which is not even in G_1. ↔ ‖

We generalize the following result of Sachs [1970] on p-critical graphs to (α,ω)-partitionable graphs.

LEMMA 3.2.14 If G is (α,ω)-partitionable, then $2\omega-2 \leq \|adj(x)\| \leq n-2\alpha+1$ for all nodes x, and these bounds are tight.

Proof: Choose a node x and a node y not adjacent to x, and choose S_1, S_2 in $\Gamma(G-y)$ such that x is in S_2. Thus $S_1 \cap adj(x) \neq \phi$. Choose z in $S_1 \cap adj(x)$, and observe $\theta(G-z)$. Each node of S_1 is paired with a node of S_2, except for one node of S_2, not x, which is paired with y. Thus x is paired with some other node in S_1. Therefore $\|S_1 \cap adj(x)\| \geq 2$.

There are $\omega-1$ distinct such S_1's in $\Gamma(G-y)$ where x is not in S_1, thus $\|adj(x)\| \geq 2\omega-2$. Since the complement of G is (ω,α)-partitionable, $2\omega-2 \leq \|adj(x)\| \leq n-2\alpha+1$.

Let $\alpha,\omega \geq 2$ be arbitrary integers. Then $C^{\omega-1}_{\alpha\omega+1}$ is (α,ω)-partitionable and $\|adj(x)\| = 2\omega-2$ for all nodes x. And

the complement of $C^{\alpha-1}_{\alpha\omega+1}$ is (α,ω)-partitionable with $\|adj(x)\| = n-2\alpha+1$ for all nodes x. ∎

3.3 CIRCLE GRAPHS

In this section we will show that the Berge Strong Perfect Graph Conjecture holds for the class of circle graphs. First we will introduce the key property, the Triple Property, and demonstrate some of its ramifications.

LEMMA 3.3.1 (The Triple Property) If G is an (α,ω)-partitionable circle graph, then no circle with chords that derives G contains three parallel chords (see Figure 3.3.1).

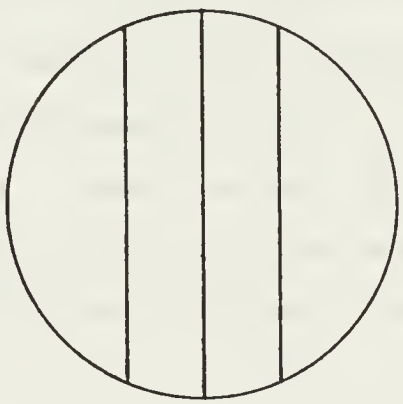


Figure 3.3.1 Three parallel chords.

Proof: Assume G is (α,ω)-partitionable and there is a circle with chords deriving G that contains three parallel chords. Removing the center chord of the three parallel chords, and all chords intersecting it, results in a circle with chords that derives G-N(x), where x corresponds to the center chord. In G-N(x) the two nodes corresponding to the other two parallel chords must be in different connected components, thus G-N(x) is disconnected. However, by Lemma 3.2.9, G-N(x) is connected. ↛ ∎

COROLLARY 3.3.2 If G is an (α,ω)-partitionable circle graph, then G is $K_{1,3}$-free. Furthermore, the Berge Strong Perfect Graph Conjecture holds for the class of circle graphs.

The result of Berge's Conjecture for the class of circle graphs follows from the class of $K_{1,3}$-free graphs (Parthasarathy and Ravindra [1976]). However, partitionable circle graphs are more than just $K_{1,3}$-free. We will give a direct proof that the Berge Strong Perfect Graph Conjecture holds for the class of circle graphs. The proof will be informative about the structure of circle graphs. Let us do so by continuing with more consequences of the Triple Property.

COROLLARY 3.3.3 If S is a stable set of an (α,ω)-partitionable circle graph G, then in any circle with chords that derives G, S corresponds to a collection of chords that go 'around' the circle (see Figure 3.3.2).

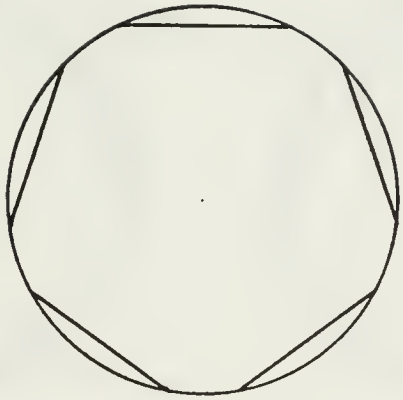


Figure 3.3.2 Chords going 'around' the circle.

COROLLARY 3.3.4 If S is an α-stable set of an (α,ω)-partitionable circle graph G and x is a node of G-S, then in any circle with chords that derives G, the chord corresponding to x intersects either i) a single chord corresponding to S, without going across the middle (see Figure 3.3.3), or ii) two consecutive chords corresponding to S.

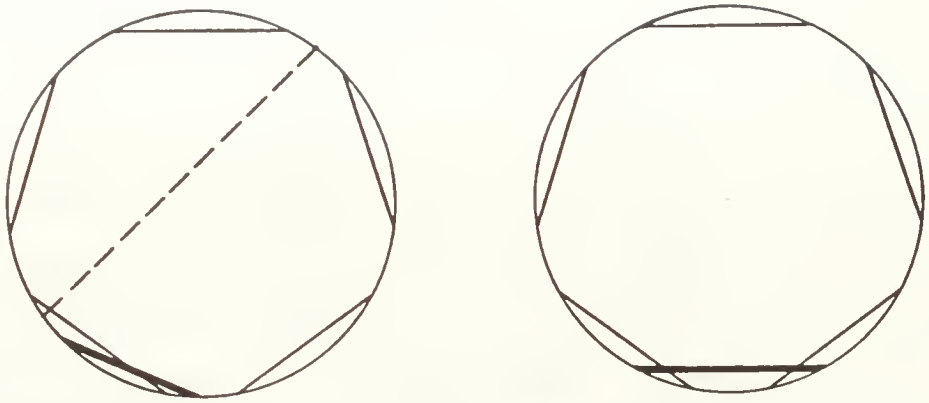


Figure 3.3.3 i) A chord intersecting one chord, without going across the middle as indicated by the bold chord (the dotted chord is forbidden); ii) A chord intersecting two consecutive chords.

COROLLARY 3.3.5 If S_1, S_2 are disjoint sets of $\Gamma(G-x)$ for any node x of an (α,ω)-partitionable circle graph G, then, in any circle with chords that derives G, $S_1 \cup S_2$ corresponds to a collection of chords that forms either i) a chordless path going 'around' the circle (see Figure 3.3.4), or ii) a chordless cycle.

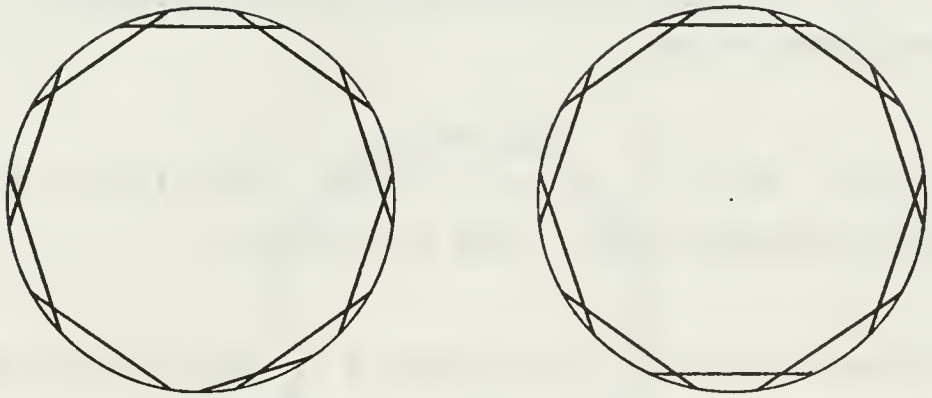


Figure 3.3.4 A chordless path going 'around' the circle, and a chordless cycle.

Proof: By Corollary 3.2.11 $S_1 \cup S_2$ is connected. Hence α-1 chords corresponding to S_2 must intersect two chords of S_1. The last chord of S_2 could intersect only one chord of S_1, thereby forming a chordless path going 'around' the circle, or could intersect two chords of S_1, thereby forming a chordless cycle. ∎

COROLLARY 3.3.6 If G is an (α,ω)-partitionable circle graph, then $\|adj(x)\| = 2\omega-2$ for all nodes x.

Proof: Choose a node x and a node y not adjacent to x. By the Triple Property, $\|S \cap adj(x)\| < 2$ for each S in $\Gamma(G-y)$. Since ω-1 S's of $\Gamma(G-y)$ account for all of adj(x), $\|adj(x)\| < 2\omega-2$. By Lemma 3.2.14 $\|adj(x)\| > 2\omega-2$. Therefore $\|adj(x)\| = 2\omega-2$. ∎

THEOREM 3.3.7 If G is an (α,ω)-partitionable circle graph, then G contains an odd chordless cycle of size > 5 or the complement of one.

Proof: If $\alpha = 2$ or $\omega = 2$, then this follows for any (α,ω)-partitionable graph. Thus assume $\alpha,\omega > 3$.

Choose a node x in G and choose S in $\Gamma(G-x)$. Since S is a maximum stable set, x is adjacent to S, yet by the Triple Property $\|S \cap adj(x)\| < 2$. Since there are ω such S's and by Corollary 3.3.6 $\|adj(x)\| = 2\omega-2$, there exist S_1, S_2 in $\Gamma(G-x)$ such that $\|S_1 \cap adj(x)\| = \|S_2 \cap adj(x)\| = 1$ and $\omega-2$ other S's in $\Gamma(G-x)$ such that $\|S \cap adj(x)\| = 2$.

Case 1) $\omega = 3$. (Here we could appeal to Tucker's [1977] result on 3-chromatic graphs, but the following is more direct.) Choose a node x, and choose S_1, S_2 in $\Gamma(G-x)$ such that $\|S_1 \cap adj(x)\| = \|S_2 \cap adj(x)\| = 1$. By Corollary 3.3.5, in any circle with chords deriving G, there are only two possible chord configurations for $S_1 \cup S_2$.

Subcase 1.i) $S_1 \cup S_2$ corresponds to a collection of chords that forms a chordless path going 'around' the circle. By Theorem 3.2.13 $S_1 \cup S_2 \cup x$ is biconnected, hence the chord corresponding to x must intersect the two end chords of the path. Since $\|(S_1 \cup S_2) \cap adj(x)\| = 2$, these are the only

intersections for this chord. There is only one way to do this (see Figure 3.3.5).

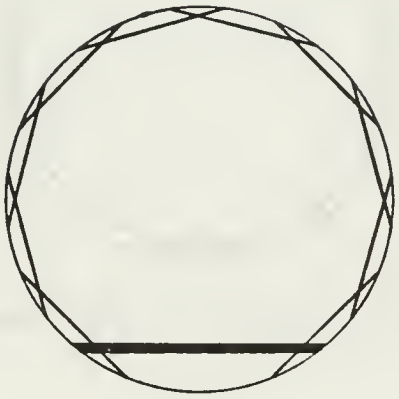


Figure 3.3.5 $S_1 \cup S_2 \cup x$ for subcase 1.i.

This results in $S_1 \cup S_2 \cup x$ being an odd chordless cycle of size $2\alpha+1$, which is > 7. Therefore G contains an odd chordless cycle of size > 5.

Subcase 1.ii) $S_1 \cup S_2$ corresponds to a collection of chords that forms a chordless cycle. By the Triple Property, there is only one way, up to symmetry, that the chord corresponding to x could intersect $S_1 \cup S_2$ such that $\|S_1 \cap adj(x)\| = \|S_2 \cap adj(x)\| = 1$ (see Figure 3.3.6).

Figure 3.3.6 $S_1 \cup S_2 \cup x$ for subcase 1.ii.

Consider the third α-stable set, S_3, in $\Gamma(G-x)$ which must intersect adj(x) twice. Choose y in $S_3 \cap$ adj(x). Since $S_3 \cup (S_1 \cup S_2 \cup x) = G$, the adj(y) is contained in $S_1 \cup S_2 \cup x$ and by Corollary 3.3.6 is of size $2\omega-2 = 4$. There is only one way, up to symmetry, that the chord corresponding to y could intersect $S_1 \cup S_2 \cup x$ without violating the Triple Property and keeping $\omega = 3$ (see Figure 3.3.7).

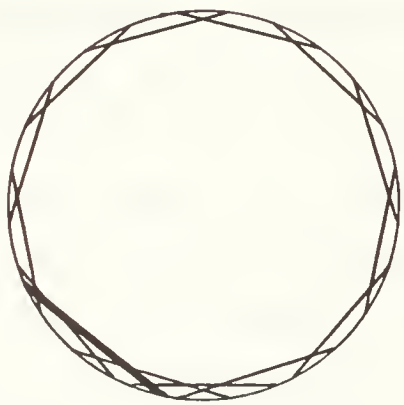


Figure 3.3.7 $S_1 \cup S_2 \cup x \cup y$.

Similarly, the chord corresponding to the other node z in $S_3 \cap$ adj(x) is uniquely placed (see Figure 3.3.8).

Figure 3.3.8 $S_1 \cup S_2 \cup x \cup y \cup z$.

Removing the two chords corresponding to $(S_1 \cup S_2) \cap adj(x)$ and the two adjacent chords in $S_1 \cup S_2$ results in an odd chordless cycle of size $2\alpha-1$, which is > 5. Therefore G contains an odd chordless cycle of size > 5.

Case 2) $\omega > 4$. Choose a node x, and choose S_1, S_2 in $\Gamma(G-x)$ such that $\|S_1 \cap adj(x)\| = \|S_2 \cap adj(x)\| = 2$. Again, by Corollary 3.3.5, there are two possibilities to consider.

Subcase 2.i) $S_1 \cup S_2$ corresponds to a collection of chords that forms a chordless path going 'around' the circle. Again, by Theorem 3.2.13, $S_1 \cup S_2 \cup x$ is biconnected, hence the chord corresponding to x must intersect the two end chords of the path. Since $\|(S_1 \cup S_2) \cap adj(x)\| = 4$, this chord must intersect

only the last two chords on both ends of the path (see Figure 3.3.9).

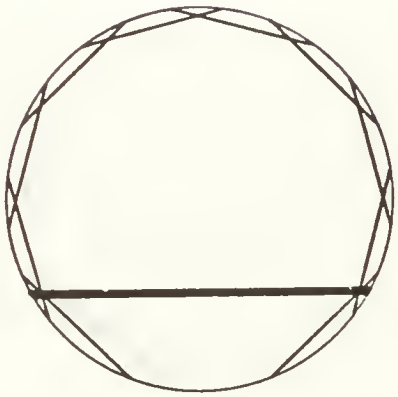


Figure 3.3.9 $S_1 \cup S_2 \cup x$ for subcase 2.i.

Removing the end chords of $S_1 \cup S_2$ results in an odd cycle of size $2\alpha-1$, which is > 5. Therefore G contains an odd chordless cycle of size > 5.

Subcase 2.ii) $S_1 \cup S_2$ corresponds to a collection of chords that forms a chordless cycle. Again, by the Triple Property, there is only one way, up to symmetry, that the chord corresponding to x could intersect $S_1 \cup S_2$ such that $\|S_1 \cap adj(x)\| = \|S_2 \cap adj(x)\| = 2$ (see Figure 3.3.10).

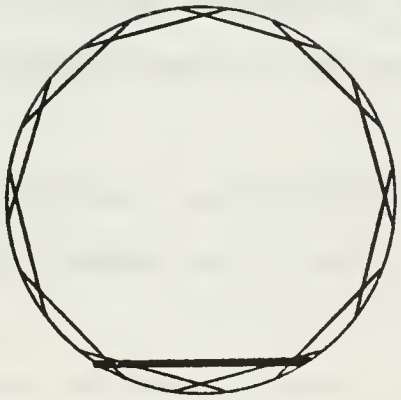


Figure 3.3.10 $S_1 \cup S_2 \cup x$ for case 2.ii.

Again, removing the two chords of $S_1 \cup S_2$, such that the cycle of chords involving x is not broken, results in an odd chordless cycle of size $2\alpha-1$, which is > 5. Therefore G contains an odd chordless cycle of size > 5.

In all cases, G contains an odd chordless cycle of size > 5 or the complement of one. ∎

COROLLARY 3.3.8 The Berge Strong Perfect Graph Conjecture holds for the class of circle graphs.

3.4 $K_{1,3}$-FREE GRAPHS

Parthasarathy and Ravindra [1976] were the first to show that the Berge Strong Perfect Graph Conjecture holds for the class of $K_{1,3}$-free graphs. In this section we give a new, very

simple proof of the conjecture for $K_{1,3}$-free graphs. We will capitalize on many properties of (α,ω)-partitionable circle graphs that carry over to (α,ω)-partitionable $K_{1,3}$-free graphs.

LEMMA 3.4.1 If G is an (α,ω)-partitionable $K_{1,3}$-free graph, then $\|adj(x)\| = 2\omega-2$ for all nodes x.

Proof: (This is nearly word for word the same proof as for Corollary 3.3.6) Choose a node x and a node y not adjacent to x. Since G is $K_{1,3}$-free, $\|S \cap adj(x)\| < 2$ for each S in $\Gamma(G-y)$. And since $\omega-1$ S's of $\Gamma(G-y)$ account for all of adj(x), $\|adj(x)\| < 2\omega-2$. By Lemma 3.2.14 $\|adj(x)\| > 2\omega-2$. Therefore $\|adj(x)\| = 2\omega-2$. ∎

COROLLARY 3.4.2 If G is an (α,ω)-partitionable $K_{1,3}$-free graph, then, for any node x, $\Gamma(G-x)$ contains

i) two α-stable sets S, such that $\|S \cap adj(x)\| = 1$, and

ii) $\omega-2$ α-stable sets S, such that $\|S \cap adj(x)\| = 2$.

Furthermore, if y is a node adjacent to x, then $\Gamma(G-x)$ contains

i) one α-stable set S, such that $\|S \cap adj(y)\| = 0$,

ii) one α-stable set S, such that $\|S \cap adj(y)\| = 1$, and

iii) $\omega-2$ α-stable sets S, such that $\|S \cap adj(y)\| = 2$.

LEMMA 3.4.3 If S_1, S_2 are two stable sets of a $K_{1,3}$-free graph such that $S_1 \cup S_2$ forms a connected graph, then $S_1 \cup S_2$ is a chordless path or a chordless cycle.

Proof: Since $S_1 \cup S_2$ is $K_{1,3}$-free, the degree of any node in $S_1 \cup S_2$ is at most 2. Thus for $S_1 \cup S_2$ to be connected, it must be a chordless path or a chordless cycle. ∎

THEOREM 3.4.4 If G is an (α,ω)-partitionable $K_{1,3}$-free graph, then G contains an odd chordless cycle of size > 5 or the complement of one.

Proof: If $\alpha = 2$ or $\omega = 2$, then this follows for any (α,ω)-partitionable graphs. Thus assume $\alpha,\omega > 3$.

Choose a node x in G. By Corollary 3.4.2 there exist S_1, S_2 in $\Gamma(G-x)$ such that $\|S_1 \cap adj(x)\| = \|S_2 \cap adj(x)\| = 1$. By Corollary 3.2.11 $S_1 \cup S_2$ is connected, hence by Lemma 3.4.3 there are two cases to be considered for $S_1 \cup S_2$.

Case i) $S_1 \cup S_2$ is a chordless path. By Theorem 3.2.13 $S_1 \cup S_2 \cup x$ is biconnected, hence x must be adjacent to the endpoints of the path. Since $\|(S_1 \cup S_2) \cap adj(x)\| = 2$, these are the only adjacencies for x. This results in $S_1 \cup S_2 \cup x$ being an odd chordless cycle of size $2\alpha+1$, which is > 7. Therefore G contains an odd chordless cycle of size > 5.

Case ii) $S_1 \cup S_2$ is a chordless cycle. Choose u in $S_1 \cap adj(x)$ and v in $S_2 \cap adj(x)$. If u and v are not adjacent, then each path connecting u and v in $S_1 \cup S_2$ together with x would form a chordless cycle of size > 5.

Assume u and v are adjacent. By Theorem 3.2.3 there exists an ω-clique C_1 such that $C_1 \cap S_1 = \phi$, and thus u is not in C_1 and $C_1 \cap S_2 \neq \phi$. By Corollary 3.2.5 x is in C_1, thus with v being the only adjacency of x in S_2, $C_1 \cap S_2 = \{v\}$. Choose z_1 in C_1 not adjacent to u. z_1 exists otherwise $C_1 \cup u$ is a clique of size $\omega+1$. Thus z_1 is adjacent to x and v and not u (see Figure 3.4.1).

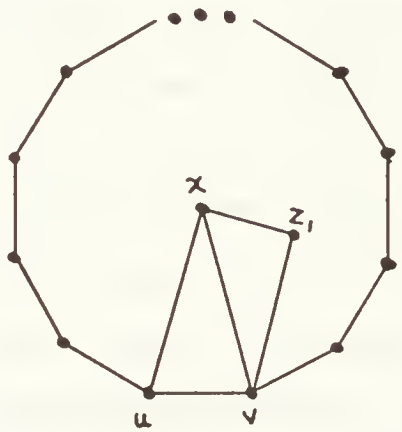


Figure 3.4.1 $S_1 \cup S_2 \cup x$ with the z_1 adjacencies of x and v.

Since G is $K_{1,3}$-free, $\|x \cup (S_1 \cap adj(z_1))\| < 2$. Yet with z_1 not in S_1, $\|S_1 \cap adj(z_1)\| > 1$, and thus $\|S_1 \cap adj(z_1)\| = 1$. Hence, by Corollary 3.4.2, $\|S_2 \cap adj(z_1)\| = 2$. This implies that z_1 is adjacent to exactly one other node of S_1 and one of S_2 not yet pictured in Figure 3.4.1. Designate these nodes as u_1 and v_1, respectively.

Locate u_1 on the cycle of $S_1 \cup S_2$. If v_1 is not on the path from u_1 to u in $S_1 \cup S_2$ that avoids v, then this path

together with x and z_1 would be an odd chordless cycle of size > 5.

Assume v_1 is on this path. If v and u_1 are not adjacent, or similarly u_1 and v_1 are not adjacent, then the path that connects these two in $S_1 \cup S_2$ that avoids u together with z_1 would be an odd chordless cycle of size > 5.

Assume v and u_1, and also u_1 and v_1, are adjacent. Thus we have the graph shown in Figure 3.4.2.

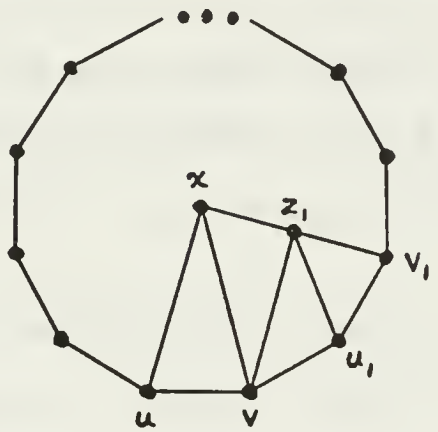


Figure 3.4.2 $S_1 \cup S_2 \cup x \cup z_1$.

Similarly, obtain clique C_2 such that $C_2 \cap S_2 = \phi$, and locate z_2 in C_2 not adjacent to v. Hence z_2 and z_1 are distinct. Find the rest of the adjacencies of z_2 denoting them as u_2 in S_1 and v_2 in S_2. The analogous argument holds, and either an odd chordless cycle has already been located or we have the graph shown in Figure 3.4.3, where u_2, v_2, u, v, u_1 and v_1 are distinct since $\alpha > 3$, and u_2 and v_1 are adjacent iff

$\alpha = 3$. z_1 and z_2 are not adjacent since $u_2 \cup u \cup z_1$ is a 3-stable set and G is $K_{1,3}$-free.

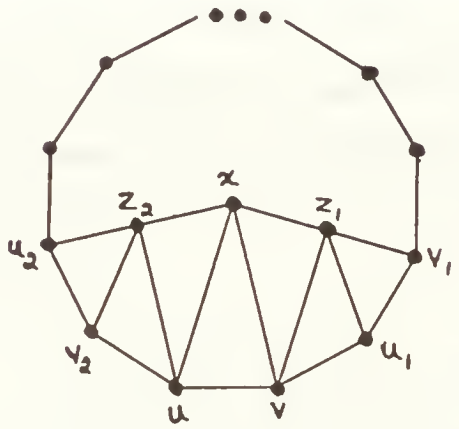


Figure 3.4.3 $S_1 \cup S_2 \cup x \cup z_1 \cup z_2$.

But alas, the path from v_1 to u_2 in $S_1 \cup S_2$ avoiding v together with z_2, x and z_1 forms an odd chordless cycle of size $2\alpha-1$, which is > 5.

Therefore G contains an odd chordless cycle of size > 5 or the complement of one. ∎

COROLLARY 3.4.5 The Berge Strong Perfect Graph Conjecture holds for the class of $K_{1,3}$-free graphs.

CHAPTER 4: ALGORITHMIC ASPECTS

Gavril [1973] was the first to show that the problems of finding a maximum weighted clique and a maximum weighted stable set of a circle graph, given a corresponding circle with chords, are both polynomial. In this chapter we shall give very efficient algorithms for these problems. But first we shall show the relation between a circle with chords and the sorting of a permutation on a system of parallel stacks mentioned in the introduction. Even and Itai [1971] were the first to show this relation. In our treatment we have changed the notation and consolidated some of the original ideas (see Golumbic [1980]). Tarjan [1972] gives a general discussion on the properties of sorting using a network of queues and stacks.

4.1 SORTING ON A SYSTEM OF PARALLEL STACKS

Consider the sorting process on a system of parallel stacks as going from right to left and consider the permutation [2 8 3 5 6 1 7 4]. To sort the permutation we take the first element 2 and place it on the first stack. The 8 must go on a second stack since the 2 must be available as soon as the 1 is processed. The 3 can be placed on the 8, and the 5 must go on a third stack, with the 6 on a fourth. Figure 4.1.1 shows the current situation.

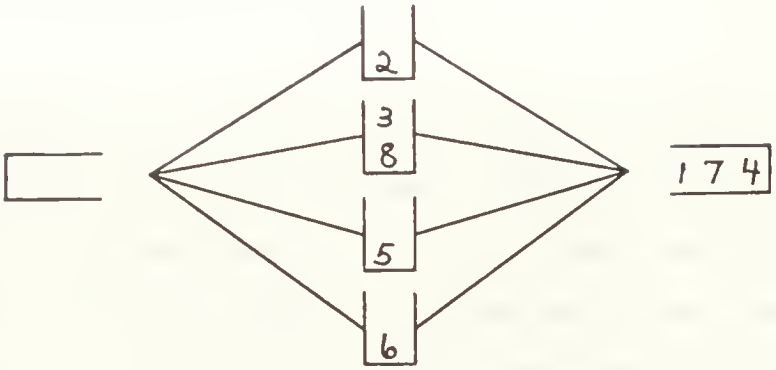


Figure 4.1.1 A partial sort of [2 8 3 5 6 1 7 4].

The 1 can be placed on any stack and immediately passed on to the left. After which the 2 can also be moved left clearing a stack. The 3 can then also be moved left. See Figure 4.1.2.

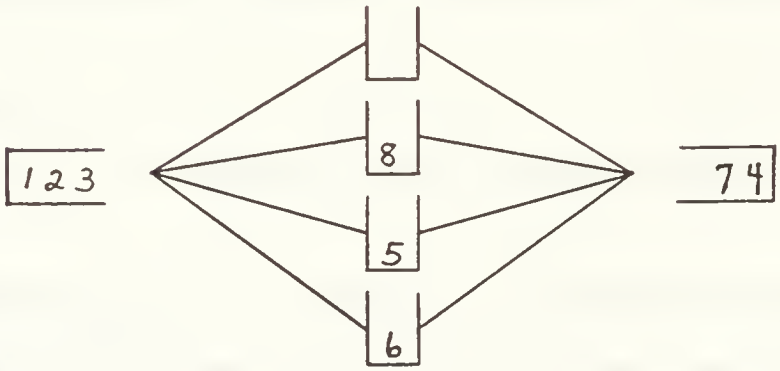


Figure 4.1.2 A partial sort of [2 8 3 5 6 1 7 4].

The 7 can go on the 8, after which the 4 passes through to the left allowing the 5,6,7 and 8 to also continue on.

In the placing of elements on the stacks a choice needs to be made as to which stack to use for each element. It would be good to find an optimal assignment, one that uses the fewest number of stacks, for the elements of the permutation. A greedy algorithm, one that does not start a new stack until necessary, will not work. Consider the permutation [5 2 4 1 6 3]. A greedy algorithm would place 5 on stack 1, 2 on stack 1, 4 on stack 2, 1 passes through and then also 2. 6 is placed on stack 3, 3 passes through and the rest follow. Three stacks were used. But consider the following: place 5 on stack 1, 2 on stack 2, 4 on stack 1, 1 passes through and then also 2, clearing a stack! 6 is placed on stack 2, 3 passes through and the rest follow. Only two stacks were used.

Consider $H(\pi)$, the conflict graph of a permutation π, where the nodes of H are the elements of the permutation and (i,j) is an edge of H iff i and j of the permutation must be placed on two different stacks when sorting. WLOG assume $i < j$. Formally then, (i,j) is an edge of H iff there exists $\ell < i < j$ such that $pos(i) < pos(j) < pos(\ell)$, where pos(x) is the inverse permutation of π. A coloring of the graph H corresponds to an assignment of the elements of π to stacks, where each stack corresponds to a color. Hence a minimum coloring of H would give us an optimal sorting assignment of π.

We shall show that $H(\pi)$, for any π, is a circle graph. In particular, we shall transform any π into a circle graph sequence such that $H(\pi)$ is isomorphic to the graph derived from the circle graph sequence. Consider a permutation π. Begin constructing a circle graph sequence by having π be the left endpoints of the intervals. Now all we need to do is find the appropriate positions in the sequence for the right endpoints. We claim that placing the right endpoints such that they are in increasing order and as far left as possible in the sequence, and still remain right endpoints, is sufficient.

ALGORITHM 4.1.1 Construct a circle graph sequence from a permutation.

Input: a permutation π.

Output: a circle graph sequence s.

Method: Let s= π

```
        Comment:  Insert the right endpoints into s.
        Process s from left to right:
        For i= 1 to n
          If i has not been passed in s then skip to it
          Add i immediately following the current position
            in s
        Next i
```

EXAMPLE 4.1.2 Apply Algorithm 4.1.1 to the permutation [2 8 3 5 6 1 7 4]. For i=1 the algorithm skips to 1 and then adds a 1 resulting in [2 8 3 5 6 1 1 7 4]. For i= 2 and 3 no

skipping occurs and the result is [2 8 3 5 6 1 1 2 3 7 4]. For i=4 the algorithm skips to the end and adds a 4, along with 5,6,7 and 8 for i= 5,6,7 and 8, resulting in [2 8 3 5 6 1 1 2 3 7 4 4 5 6 7 8].

LEMMA 4.1.3 Algorithm 4.1.1 creates a circle graph sequence whose derived graph is isomorphic to $H(\pi)$ of the inputted permutation π.

Proof: Clearly the nodes of the derived graph of the outputted circle graph sequence correspond to the nodes of $H(\pi)$ of the inputted permutation π.

Choose an edge (i,j) of $H(\pi)$ of the inputted permutation π. WLOG assume $i < j$. Then there exists $\ell < i < j$ such that $pos(i) < pos(j) < pos(\ell)$. After the loop for $i=\ell$ the current position in s is beyond both i and j, hence the right endpoints of i and j are inserted after the left endpoints of i and j, i being first, resulting in intervals i and j overlapping. Therefore (i,j) is an edge of the graph derived from the outputted circle graph sequence.

Choose an edge (i,j) of the graph derived from the outputted circle graph sequence. WLOG assume $i < j$. Since the right endpoint of i was inserted before the right endpoint of j, and thus past the left endpoint of j, $pos(i) < pos(j)$ and there exists $\ell < i < j$ such that $pos(i) < pos(j) < pos(\ell)$.

Therefore (i,j) is an edge of $H(\pi)$ of the inputted permutation π. ∎

LEMMA 4.1.4 Algorithm 4.1.1 can be implemented in O(n) time, where n is the length of the inputted permutation.

Proof: Use an auxiliary array to keep track of the passed left endpoints. Then checking for i being skipped is O(1), making each pass of the loop O(1). The loop executes n times. ∎

See section A.2 for a specific implementation.

The converse, that every circle graph is isomorphic to $H(\pi)$ for some π, is not true. This is obvious since nodes 1 and $\pi(n)$ are isolated in $H(\pi)$ and not every circle graph has isolated nodes. However, it is true that every circle graph, after adding a few isolated nodes, is isomorphic to $H(\pi)$ for some π. In particular, we shall show how to transform any circle graph sequence s, after adding intervals with adjacent endpoints, into a permutation π such that $H(\pi)$ is isomorphic to the derived graph of s + these added intervals. Our goal in adding intervals will be to create a circle graph sequence whose left endpoints will give us the desired permutation. This is just the reverse procedure of Algorithm 4.1.1. The first thing to note is that the graph derived from a circle graph sequence is invariant under renumbering of the intervals,

but the resulting left endpoint sequence is not. So even after adding some appropriate intervals with adjacent endpoints, we will need to renumber. We claim that it suffices to renumber, such that the right endpoints are in increasing order. Again, this is simply the reverse procedure of Algorithm 4.1.1.

To determine the appropriate added intervals, consider the circle graph sequence [1 2 3 4 5 6 6 1 3 7 4 5 7 2]. Interval 1 overlaps 2,3,4 and 5. In the resulting permutation we will need, for each of i= 2,3,4 and 5, some ℓ such that $\ell < 1 < i$ and $pos(1) < pos(i) < pos(\ell)$. Of course 1,2,3,4 and 5 will be renumbered in the final result, but we will still need the ℓ's. We claim that it suffices to add intervals with adjacent endpoints so that the resulting circle graph sequence contains an interval with adjacent endpoints--either an original interval or a newly added one--between every left and right endpoint, where the left endpoint is to the left of the right endpoint. For example, for the above sequence we would generate [1 2 3 4 5 6 6 1 3 7 x x 4 5 6 2]. The x's represent the added interval. Note that the interval 6 is between the left endpoint of 5 and the right endpoint of 1 so none need be added there, and that the left endpoint of 7 is not to the left of the right endpoint of 3 so none need be added there either. Between the left endpoint of 7 and the right endpoint of 4 one did need to be added.

ALGORITHM 4.1.5 Construct a permutation from a circle graph sequence.

Input: a circle graph sequence s.

Output: a permutation π.

Method: Let p= s

Comment: Insert the appropriate intervals.

Process p from left to right:

For every right endpoint in p, that is not of an interval with adjacent endpoints

If there is no interval with adjacent endpoints between this right endpoint and the nearest left endpoint to the left then add one immediately prior to this right endpoint

Next right endpoint

Renumber the intervals of p such that the right endpoints are in increasing order

Let π = the sequence of left endpoints of p

EXAMPLE 4.1.6 Consider the circle graph sequence [1 2 3 4 5 6 6 1 3 7 4 5 7 2]. Algorithm 4.1.5 first examines the right endpoint of 6, but this is of an interval with adjacent endpoints. The right endpoints of 1 and 3 both have the interval 6 between them and the nearest left endpoint to the left. The right endpoint of 4 has no interval with adjacent endpoints between it and the left endpoint of 7, so one is added resulting in [1 2 3 4 5 6 6 1 3 7 x x 4 5 7 2]. The final right endpoints of 5,7 and 2 are satisfied by the newly

added interval. The renumbering gives [2 8 3 5 6 1 1 2 3 7 4 4 5 6 7 8]. Thus π = [2 8 3 5 6 1 7 4].

LEMMA 4.1.7 Algorithm 4.1.5 creates a permutation π such that $H(\pi)$ is isomorphic to the graph derived by the inputted circle graph sequence plus some isolated nodes.

Proof: Clearly the nodes of $H(\pi)$ of the outputted permutation π correspond to the nodes of the graph derived by the inputted circle graph sequence plus some isolated nodes.

Choose an edge (i,j) of the graph derived from the inputted circle graph sequence. WLOG assume the left endpoint of i is to the left of the left endpoint of j. The loop of Algorithm 4.1.5, since i and j are intervals without adjacent endpoints, insures that between the right endpoint of i and the left endpoint of j there is another interval ℓ with adjacent endpoints. Hence the renumbering R will produce numbers such that in the resulting permutation $R(\ell) < R(i) < R(j)$ and $pos(R(i)) < pos(R(j)) < pos(R(\ell))$. Thus (R(i),R(j)) is an edge of $H(\pi)$ of the outputted permutation π.

Choose an edge (i,j) of $H(\pi)$ of the outputted permutation π. WLOG assume $i < j$. Then there exists $\ell < i < j$ such that $pos(i) < pos(j) < pos(\ell)$. Therefore in the renumbered circle graph sequence i and j must overlap, and hence neither can be an added interval. Thus, for RI being the inverse renumbering,

(RI(i),RI(j)) is an edge of the graph derived from the inputted circle graph sequence. ∎

LEMMA 4.1.8 Algorithm 4.1.5 can be implemented on O(n) time, where n is the length of the inputted circle graph sequence.

Proof: Use a flag to keep track of a left endpoint occurring, without an interval with adjacent endpoints being found to the right of it, as the algorithm passes over the input from left to right. Then in a single scan of the input the added intervals can be placed by checking this flag at each right endpoint. This costs O(n). Use an auxiliary array to point at the right endpoints while renumbering. Since the number of intervals is at most doubled by the adding process, renumbering is O(n). The final pass to find π is also O(n). ∎

See section A.2 for a specific implementation.

COROLLARY 4.1.9 The assignment of stacks during the sorting of a permutation on a system of parallel stacks is polynomially equivalent to the coloring of a circle graph.

Proof: This follows immediately from Algorithms 4.1.1 and 4.1.5 and the fact that isolated nodes do not affect the coloring of graphs. ∎

THEOREM 4.1.10 (Garey, Johnson, Miller and Papadimitriou [1979]) The circle graph coloring problem is NP-complete.

Proof: See the literature; Garey, Johnson, Miller and Papadimitriou [1979]. ∎

COROLLARY 4.1.11 The problems of finding an optimal assignment for sorting a permutation on a system of parallel stacks, and of finding an optimal construction of a connection board, where connections are allowed on every side, are NP-complete.

4.2 GENERATING EDGES

Recall that a circle graph sequence can be abstractly defined as a finite sequence of integers, where every integer occurs exactly twice. It was originally defined to be a sequence of numbers taken from a circle with chords, where the numbers labeled the endpoints of the chords with matching numbers at the endpoints of a common chord. Recall also that a circle graph sequence can be seen to be a representation of a collection of intervals where the endpoints of the intervals are all distinct. Each interval corresponds to the two occurrences of an integer in the sequence.

Let us define three graphs that can be derived from a circle graph sequence s, each having as nodes the integers of the sequence:

i) G(s), the overlap graph; (i,j) is an edge iff intervals i and j overlap one another, (this is what was previously specified as the graph derived from a circle graph sequence)

ii) C(s), the containment graph; (i,j) is an edge iff interval i is contained in interval j, or vice versa, and

iii) D(s), the disjoint graph; (i,j) is an edge iff intervals i and j are disjoint.

Clearly the edges of G(s), C(s) and D(s) are disjoint and their union contains all possible unordered pairs (i,j), where $i \neq j$.

In many applications it is not necessary to explicitly generate the edges for these graphs since this information is readily obtainable from the circle graph sequence. However, it is instructive to see how the edges can be efficiently generated.

Given a circle graph sequence s, consider the unordered pair (i,j) where $i \neq j$. WLOG assume the right endpoint of i is to the left of the right endpoint of j. If (i,j) is an edge of G(s), then in s the element j occurs only once before the right endpoint of i, and that being after the left endpoint of i. If (i,j) is an edge of C(s), then in s the element j occurs again only once before the right endpoint of i, but this time also before the left endpoint of i. If (i,j) is an edge of D(s),

then in s the left endpoint of j is to the right of both endpoints of i. If, while processing s in a left to right manner, we can keep track of the order of unmatched left endpoints found so far, then edges of G(s) and C(s) can be generated upon encountering a right endpoint. And similarly, if we can keep track of the intervals to which matched endpoints have been found, then edges of D(s) can be generated upon encountering a left endpoint. We claim that this all can be done simply by using two stacks, where one keeps the information for unmatched left endpoints and the other for matched endpoints.

ALGORITHM 4.2.1 Construct the edges of G(s), C(s) and D(s) from a circle graph sequence.

Input: a circle graph sequence s.

Output: edges for G(s), C(s) and D(s).

Method: Let A and B be two empty stacks

```
        Process s from left to right:
        For each element i of s
          If i is not in A
            Then Add i to the top of A
                 For each j in B generate the edge (i,j)
                     for D(s)
            Else For each j above i in A generate the edge
                     (i,j) for G(s)
                 For each j below i in A generate the edge
                     (i,j) for C(s)
```

Remove i from A and add it to the top of B

Next element

EXAMPLE 4.2.2 Consider the circle graph sequence s= [1 2 1 3 4 3 4 2]. Following Algorithm 4.2.1: 1 and 2 are placed on A, 1 is placed on B generating (1,2) for G(s), 3 and 4 are placed on A generating (1,3) and (1,4) for D(s), 3 is placed on B generating (3,4) for G(s) and (2,3) for C(s), 4 is placed on B generating (2,4) for C(s), and 2 is placed on B. The result is edges (1,2) and (3,4) for G(s), (2,3) and (2,4) for C(s), and (1,3) and (1,4) for D(s).

LEMMA 4.2.3 Algorithm 4.2.1 creates the edges of G(s), C(s) and D(s) of the inputted circle graph sequence s.

Proof: Every edge that is generated belongs in the designated graph. And every edge that belongs to each graph will at one time or another be generated once for that particular graph. ∎

LEMMA 4.2.4 Algorithm 4.2.1 can be implemented in O(e + c + d + n) time, where e, c and d= # of edges of G(s), C(s) and D(s), respectively, for the inputted circle graph sequence s, and n= length of s. Furthermore, Algorithm 4.2.1 can be easily modified to generate any one, or two, of G(s), C(s) or D(s), with the accompanying reduction in complexity.

Proof: The outer loop of Algorithm 4.2.1 is executed O(n) times with each respective inner loop costing an additional O(e), O(c) and O(d) for the edges of G(s), C(s) and D(s), respectively. Removing the appropriate edge generating loops achieves the furthermore. ∎

See section A.3 for a specific implementation.

4.3 AN EFFICIENT CLIQUE ALGORITHM

Gavril [1973] demonstrated that the problem of finding a maximum clique in a circle graph, where the circle with chords, or equivalently the overlapping intervals, are known, can be done in polynomial time. He indicated an $O(n^3)$ method, where n is the number of intervals. We shall demonstrate an $O(e*\log_2\omega)$ algorithm for the unweighted clique problem and an $O(\delta e)$ algorithm for the weighted case, where e= # of edges of G(s), for the circle graph sequence s, $\omega = \omega(G(s))$ and δ = maximum degree of any node in G(s). The overlapping intervals are assumed to be known and a corresponding circle graph sequence will be the input to our algorithms.

Recall from chapter 1, Lemma 2.2.4, that any collection of intervals deriving a clique of size p must be of the form [1 2 ... p 1 2 ... p]. For a circle graph sequence s to derive a graph containing a clique of size p, this sequence must be a

subsequence of s, although possibly renumbered. Our goal is to take a circle graph sequence and find the largest p such that [1 2 ... p 1 2 ... p] is a subsequence. We claim, since such a sequence must be in the neighborhood of some interval, that it suffices to process the neighborhoods of each of the intervals of the inputted circle graph sequence, beginning with the interval having the leftmost right endpoint, throwing it away and continuing on to the interval having the next leftmost right endpoint, remembering the best clique found as we go along.

ALGORITHM 4.3.1 Find the maximum clique of the overlap graph of a circle graph sequence.

Input: a circle graph sequence s.

Output: $\omega(G(s))$.

Method: For i= interval having the leftmost right endpoint
remaining in s
Generate the subsequence s' of s such that
G(s')= N(i) in G(s)
Find the largest pattern of [1 2 ... p 1 2 ... p]
in s'
Delete i from s remembering the largest p obtained
so far
Next interval
Let $\omega(G(s))$= the largest p found

There are some major details yet to be handled. Given the s' generated in each loop of Algorithm 4.3.1, how can the largest pattern of [1 2 ... p 1 2 ... p] be found? s' has one very important property: Being the neighborhood of the interval of the leftmost right endpoint i, all other right endpoints of s' are to the right of the right endpoint of i. Therefore simply renumber the left endpoints of s' so that they are in increasing order and search the right endpoints for the longest increasing subsequence.

There is an efficient method for finding a longest increasing subsequence of a sequence q. The same situation arises in the problem of finding a maximum clique of a permutation graph (Even, Pnueli and Lempel [1972]). Fredman [1975] gives a complete analysis of this method. Consider a sequence of stacks. Place the first element of q on the first stack. If the second element of q is smaller than the first, place it on top of the first, otherwise start a second stack, which will also indicate that a subsequence of length two has been found. We claim that by continuing to add elements of q to these stacks with no larger number placed on top of a smaller, where the first stack is used whenever possible, and if not then the second, etc., that the number of stacks ultimately used is the length of the longest subsequence of q.

ALGORITHM 4.3.2 Find the longest increasing subsequence of a sequence.

Input: a sequence q.

Output: the length, p, of the longest increasing subsequence.

Method: Process q from left to right:

For each element i of q

Place i on the first available stack and start another only if need be

Next element

Let p= the number of stacks used

EXAMPLE 4.3.3 Consider the sequence [2 4 1 5 3]. Algorithm 4.3.2 would place 2 on stack 1, 4 on stack 2, 1 on stack 1, 5 on stack 3, and 3 on stack 2. The number of stacks used is 3, where [2 4 5] is the longest increasing subsequence.

LEMMA 4.3.4 Algorithm 4.3.2 finds the longest increasing subsequence of an inputted sequence.

Proof: When an element i is placed on stack k, it is because stack k-1 has a top element less than i, which appeared prior to i in the sequence. Inductively, this top element of stack k-1 was placed there for a similiar reason. Chaining this back to the first stack we have an increasing subsequence of size k. Therefore the total number of used stacks p indicates that there is an increasing subsequence of size p.

If there is a subsequence of size p, then clearly at least p stacks will have to be used. ‖

LEMMA 4.3.5 Algorithm 4.3.2 can be implemented in $O(n*\log_2(p+1))$ time, where n is the length of the inputted sequence and p is the length of the largest increasing subsequence.

Proof: First observe that the tops of the stacks are always monotone increasing. Hence, finding the first available stack can be accomplished by a binary search, which is $O(\log_2(p+1))$. But this occurs n times. ‖

Since Algorithm 4.3.2 needs only the right endpoint sequence of a neighborhood of an interval of a circle graph sequence to determine the largest pattern of [1 2 ... p 1 2 ... p] in that neighborhood, Algorithm 4.3.1 need only provide that information instead of a circle graph sequence s' where G(s')= N(i) in G(s). However, the associated left endpoints need to be labeled in increasing order. We claim that this can all be accomplished by preprocessing the input so that all the left endpoints are labeled in increasing order and by using the same stack processing of Algorithm 4.2.1 for finding the edges of G(s).

ALGORITHM 4.3.6 Find the maximum clique of the overlap graph of a circle graph sequence.

Input: a circle graph sequence s.

Output: $\omega(G(s))$.

Method: Renumber the intervals of s such that the left endpoints are in increasing order

Let A be an empty stack

Process s from left to right:

For each element i of s

If i is not in A

Then Add i to the top of A

Else For the collection of left endpoints above i in A, including i, generate the right endpoint sequence and pass this to Algorithm 4.3.2

Delete i from A remembering the largest p obtained from Algorithm 4.3.2 so far

Next element

Let $\omega(G(s))$= the largest p found

EXAMPLE 4.3.7 Consider the circle graph sequence [1 2 3 1 4 4 5 6 3 2 5 6], which Algorithm 4.3.6, after renumbering, leaves the same. Following Algorithm 4.3.6: 1,2 and 3 are placed on A; 1 is removed from A after generating the right endpoint sequence of 1,2 and 3, which is [1 3 2], and passing this to Algorithm 4.3.2 which returns a 2; 4 is placed on A; 4 is removed from A after passing [4] and receiving a 1; 5 and 6

are placed on A; 3 is removed from A after passing [3 5 6] and receiving a 3; 2 is removed from A after passing [2 5 6] and receiving a 3; 5 is removed from A after passing [5 6] and receiving a 2; and 6 is removed after passing [6] and receiving a 1. Therefore the maximum clique is of size 3 which comes from the [3 5 6 3 5 6], or [2 5 6 2 5 6], subsequence.

LEMMA 4.3.8 Algorithm 4.3.6 finds the maximum clique of the overlap graph of an inputted circle graph sequence s.

Proof: The stack process is the same as in Algorithm 4.2.1 for G(s). So by its correctness, and that of Algorithm 4.3.2, Algorithm 4.3.6 finds a clique.

Choose the leftmost interval i in s of a maximum clique. When Algorithm 4.3.6 encounters the right endpoint of i, the left endpoints above i in A will contain the left endpoints of the other intervals belonging to the maximum clique and possibly more, listed in order from bottom to top. Hence, when Algorithm 4.3.2 receives the right endpoint sequence, the maximum clique will be identified. ∎

Generating the right endpoint sequence of a collection of intervals, as is needed in Algorithm 4.3.6, is not a very efficient thing since the entire sequence of s to the right of the leftmost right endpoint must be scanned. However, finding the right endpoint position sequence can be done efficiently

and is equivalent, in terms of an increasing subsequence, to the right endpoint sequence. For example, given the circle graph sequence of Example 4.3.7, [1 2 3 1 4 4 5 6 3 2 5 6], the right endpoint sequence of 1,2 and 3 is [1 3 2] and the right endpoint position sequence is [4 10 9]. Two right endpoints are in order iff the two right endpoint positions are in order. By having Algorithm 4.3.6 do a preprocessing for right endpoints, an O(n) time operation, the right endpoint position sequence can be calculated in O(length of the right endpoint sequence) time. This sequence is passed to Algorithm 4.3.2 which is not affected by the magnitude of the inputted sequence elements. In this case the renumbering of s in Algorithm 4.3.6 is not necessary. It is also sufficient to pass to Algorithm 4.3.2 the right endpoint position sequence of only the adjacency set instead of the neighborhood, and then add 1 to the returned value.

LEMMA 4.3.9 Algorithm 4.3.6 can be implemented in $O(e*\log_2\omega + n)$ time, where e= # of edges of G(s), for the inputted circle graph sequence s, $\omega = \omega(G(s))$ and n= the length of s.

Proof: Preprocessing s for the right endpoint positions is O(n). The complexity of generating the right endpoint position sequences is the same as that of generating the edges of G(s), O(e + n). Now we must calculate the cost of using Algorithm 4.3.2.

Algorithm 4.3.2 is called n times, each call processing a sequence no larger than the size of the corresponding adjacency set. All together that costs $O(\sum O(\|adj(i)\| * \log_2(p_i+1)))$, where p_i is no larger than the size of the maximum clique containing i, minus 1. Clearly this is bounded by $O(e*\log_2\omega)$. ∎

The actual clique itself can be carried along via a sequence of pointers, where one pointer is added each time an element is processed in Algorithm 4.3.2. Hence, the clique can be explicitly obtained at no additional complexity. See section A.4 for a specific implementation.

Consider now the weighted clique problem of an overlap graph of a circle graph sequence. Algorithm 4.3.6 is quite adequate for this purpose, however Algorithm 4.3.2 is not. For example, let s be the circle graph sequence [1 2 3 4 5 1 4 2 3 5] with the weight of 1 for all but 4, which has the weight of 3. Algorithm 4.3.6, when processing the right endpoint of 1, generates the right endpoint sequence of [1 4 2 3 5], passing this to Algorithm 4.3.2, which in turn generates the stack situation shown in Figure 4.3.1.

Figure 4.3.1 The final stack situation of Algorithm 4.3.2 after processing [1 4 2 3 5].

The algorithm needs to return the weight of 5, which comes from the [1 4 5] increasing subsequence, and not 4, the number of used stacks. Essentially, Algorithm 4.3.2 would need to check, for each newly added element to the stacks, all elements currently on the stacks less than that number. For example, when stacking the 5, the fact that 4 is buried in stack 2 and being less than 5 needs to be checked. In this case the [1 4 5] subsequence would be found. Having Algorithm 4.3.2 do all this checking is not necessary since a different and more direct procedure is available.

Consider a sequence of numbers each having an assigned weight. Our goal is to find a maximum weighted increasing subsequence. Our intermediate goal is to assign to each element of the sequence a number representing the weight of the maximum weighted increasing subsequence that has been found so far, ending with that element. Hence, begin each element with its own weight. Take the first element of the sequence and for each subsequent larger element determine a new subsequence where the subsequent element is added to the end of the

subsequence of the first. If this is an improvement over what has been found so far for the subsequent element then keep it. We claim that a maximum weighted increasing subsequence can be obtained efficiently by continuing with the next element of the sequence in a similiar fashion, and then the next, etc., except that when encountering a smaller subsequent element we recurse and process this out first.

ALGORITHM 4.3.10 Find the maximum weighted increasing subsequence of a sequence.

Input: a sequence q and a weight function w.

Output: the weight, p, of the maximum weighted increasing subsequence.

Method: For each element i in q
  Let l(i)= w(i)
Next element
Process q from left to right:
For each remaining element i in q
  Call the recursive procedure proc(i)
Next element
Let p= the maximum of the l(i)

Recursive procedure proc(i)
Process from left to right:
For each remaining element j in q, to the right of i
  If i < j Then let l(j)= max(l(j), l(i)+w(j))
    Else proc(j)

Next element

Delete i from q

Return

EXAMPLE 4.3.11 Consider the sequence [1 4 2 3 5] having the weight of 1 for all but 4, which has the weight of 3. Algorithm 4.3.10 begins by assigning 1 to 1(1), 1(2), 1(3) and 1(5), and 3 to 1(4). Then proc(1) is called. Proc(1), when comparing 1 and 4, makes 1(4)= 4; 1 and 2, 1(2)= 2; 1 and 3, 1(3)= 2; 1 and 5, 1(5)= 2; and then 1 is deleted from q. Returning we get the call of proc(4), which when comparing 4 and 2, recursively calls proc(2). Proc(2) causes 1(3) and 1(5) to be 3, and then 2 is deleted from q. Returning we compare 4 and 3 and recursively call proc(3). Proc(3) causes 1(5) to be 4, and then 3 is deleted from q. Returning we compare 4 and 5 which makes 1(5)= 5. 4 is then deleted from q and in returning we find ourselves back in the main routine with only [5] as the remaining q. Calling proc(5) results in 5 being deleted from q. Thus p receives 5, the maximum of the 1(i), which comes from the [1 4 5] subsequence.

LEMMA 4.3.12 Algorithm 4.3.10 finds the maximum weighted increasing subsequence of an inputted sequence.

Proof: Clearly Algorithm 4.3.10 finds an increasing subsequence.

When Algorithm 4.3.10 calls proc(i) for i being the first element of a maximum weighted increasing subsequence, the assignment of l(j)= max(l(j), l(i)+w(j)) will be effected where j is the second element of the subsequence, giving it its maximum value. l(j) will never become larger, else replacing i, with whatever caused l(j) to become larger, yields a larger maximum weighted increasing subsequence. The second element similarly affects the label of the third element of a maximum weighted increasing subsequence, etc., and the maximum weighted increasing subsequence is identified. ∎

LEMMA 4.3.13 Algorithm 4.3.10 can be implemented in O(e+n) time, where n is the length of the inputted sequence q and e is the number of occurrences in q of i < j, for j being to the right of i.

Proof: Making the initial labels is O(n). The number of calls to proc(i) from the main routine and from proc(i) is a total of n, and the number of times the true part of the if statement in proc(i) is executed is a total of e. Finding p is also O(n). ∎

Having Algorithm 4.3.6 call Algorithm 4.3.10, instead of Algorithm 4.3.2, gives us a method for finding a maximum weighted clique of an overlap graph of a circle graph sequence. The same efficiency modifications made previously to Algorithm 4.3.6 still hold with one added criterion: the weight function

to be used in Algorithm 4.3.10 must be a composite function of the circle graph sequence and the weights, since Algorithm 4.3.10 will be receiving a right endpoint position sequence and not the right endpoint sequence. Correctness holds, but due to the increased complexity of Algorithm 4.3.10 over Algorithm 4.3.2, the complexity for finding a maximum weighted clique increases over that of the unweighted clique.

LEMMA 4.3.14 Algorithm 4.3.6, for the weighted clique problem, can be implemented in $O(\delta e + n)$ time, where e= # of edges of G(s), for the inputted circle graph sequence s, δ= the maximum degreee of the nodes of G(s) and n= the length of s.

Proof: The complexity is the same as in Lemma 4.3.9, except that Algorithm 4.3.10 is used: O(e+n) plus the cost of using Algorithm 4.3.10.

Algorithm 4.3.10 is called n times, each call processing a sequence no larger than the size of the corresponding adjacency set. All together that costs $O(\sum O(e(adj(i))+\|adj(i)\|))$, since the number of occurrences in the sequence of $i < j$, where j is to the right of i, is the number of edges in adj(i): e(adj(i)). Clearly this is bounded by $O(\delta e)$. ∎

Again, the actual clique itself can be carried along via a sequence of pointers, where one pointer is added each time a label is updated. Hence, the clique can be explicitly obtained

at no added complexity. See section A.4 for a specific implementation.

4.4 AN EFFICIENT STABLE SET ALGORITHM

Gavril [1973], in the same paper in which the maximum clique problem is discussed, demonstrated that the problem of finding a maximum stable set in a circle graph, where the circle with,clords, or equivalently the overlapping intervals, are known, can be done in polynomial time. He indicated an $O(n^3)$ method for this also, where n is the number of intervals. We shall demonstrate an O(c) algorithm for the weighted stable set problem, the unweighted problem being a special case of this, where c= # of occurrences of an interval being strictly contained in another interval in s, that is, # of edges of C(s), for the circle graph sequence s. The overlapping intervals are assumed to be known and a corresponding circle graph sequence will be the input to our algorithm.

Given a collection of intervals whose overlap graph is a stable set, it is clear that each pair of intervals must be disjoint or one must be strictly contained in the another. Such is the nature of the intervals that we want to choose from a circle graph sequence.

Consider a circle graph sequence having a weight for each interval. Scan the sequence from left to right. Our intermediate goal is to label each element of the sequence with the maximum weight of the intervals, up to and including that element, that do not overlap one another. Every time a left endpoint is encountered, it receives its label from the previous element since no new interval has been added. Every time a right endpoint of some interval i is encountered, we add to the label of the left endpoint of i the containment weight--the maximum weight of the intervals contained in i that do not overlap one another--plus the weight of i, and choose between this sum and the label from the element previous to the right endpoint of i. Calculating the containment weight of i is a recursive subproblem of the original sequence where the containment sequence of i is made up of the endpoints of the intervals completely contained in i. We claim that a maximum weighted stable set can be obtained efficiently by preprocessing the sequence for all the containment sequences, and actually calculating each containment weight only once in a recursive manner.

ALGORITHM 4.4.1 Find the maximum weighted stable set of the overlap graph of a circle graph sequence.

Input: a circle graph sequence s and a weight function w.

Output: $\alpha(G(s), w)$.

Method: Comment: Build the containment adjacencies.

Let A be an empty stack and adj be an array of empty

```
    lists
  Process s from left to right:
  For each element i of s
    If i is not in A Then Add i to the top of A
    Else For each j below i in A add j to adj(i)
         and Remove i from A
  Next element
  Comment:  Build the containment sequences.
  Let cs be an array of empty sequences
  Process s from left to right:
  For each element i of s
    For each element j of adj(i)
      Add i to cs(j)
    Next element j
  Next element i
  Comment:  Initialize the containment weights and run
            the recursive procedure.
  Let c be an array of undefined numbers
  Call the recursive procedure proc(s)
  Let α(G(s), w)= the returned value of proc(s)

Recursive procedure proc (s)
  Process s from left to right:
  For each element i of s
    If i is a left endpoint Then
         Let l(left endpoint i)= l(the previous element)
                  or 0 if there is no previous element
```

```
        Else If c(i) is undefined then let c(i)=
                    0 if cs(i) is empty,
                    else the returned value of proc(cs(i))
                Let l(right endpoint i)=
                    max{l(the previous element),
                        l(left endpoint i) + c(i) + w(i)}
            Next element
            Return l(the last element of s)
```

EXAMPLE 4.4.2 Consider the circle graph sequence [1 1 2 3 4 3 2 4] with the weight of 1 for all the intervals. The preprocessing of Algorithm 4.4.1 makes adj(1)= empty, adj(2)= empty, adj(3)= {2} and adj(4)= empty. Thus the containment sequences are cs(1)= empty, cs(2)= [3 3], cs(3)= empty and cs(4)= empty. Then the initial call of proc ([1 1 2 3 4 3 2 4]) is made: for the left endpoint of 1, l= 0; for the right endpoint of 1, since c(1) is undefined and cs(1) is empty, c(1)= 0 and l= max(0, 0+0+1)= 1; for the left endpoints of 2,3 and 4, l= 1; for the right endpoint of 3, since c(3) is undefined and cs(3) is empty, c(3)= 0 and l= max(1, 1+0+1)= 2; for the right endpoint of 2, since c(2) is undefined and cs(2)= [3 3], a recursive call of proc ([3 3]) is made (for the left endpoint of 3, l= 0; for the right endpoint of 3, since c(3)= 0, l= max(0, 0+0+1)= 1) and a 1 is returned making c(2)= 1 and l= max(2, 1+1+1)= 3; for the right endpoint of 4, since c(4) is undefined and cs(4) is empty, c(4)= 0 and l= max(3, 1+0+1)= 3.

Thus a 3 is returned and set equal to $\alpha(G(s), w)$, where 1,2 and 3 is the largest stable set.

LEMMA 4.4.3 Algorithm 4.4.1 finds the maximum weighted stable set of the overlap graph of an inputted circle graph sequence.

Proof: The stack process that creates an adj(i), where every element j indicates that the interval i is contained in j, is the same as in Algorithm 4.2.1 for C(s). Thus the correctness follows from Lemma 4.2.3. The correctness of the building of the containment sequences also follows. Clearly Algorithm 4.4.1 finds a stable set.

Given a maximum weighted stable set S, form a subset T of S such that every interval of S is in T or is contained in some interval of T. By induction on the size of the circle graph sequence s, c(i), for i in T, is calculated correctly. Hence the labeling that Algorithm 4.4.1 produces for each of the right endpoints of T, this occurring only in the topmost level of recursion, is the partial sum of the maximum weighted stable set. Therefore the labeling of the last element in s is the maximum weight. ∎

LEMMA 4.4.4 Algorithm 4.4.1 can be implemented in O(c+n) time, where c= # of occurrences in s of an interval being strictly contained in another interval, that is, # of edges of

C(s), for the inputted circle graph sequence s, and n= the length of s.

Proof: By Lemma 4.2.4 the complexity of the preprocessing is O(c+n). The recursive procedure is called once for s and once for each cs(i)--for cs(i) being empty consider the assignment of 0 to c(i) as an empty call--each call costing O(‖input‖). Thus the total cost of using proc is O(n + $\sum$ ‖cs(i)‖), which is O(c+n) since ‖cs(i)‖ is the size of adj(i) in C(s). ‖

As in the clique algorithms, the actual stable set can be carried along via a sequence of pointers, where one pointer is added each time a label is updated. Hence, the stable set can be explicity obtained at no added complexity. See section A.5 for a specific implementation.

CHAPTER 5: OPEN PROBLEMS

There is still much to be learned about circle graphs. In chapter 1 we discussed many different graphs and showed whether or not they were circle graphs, and in the event that they were, whether or not they were uniquely constructable. But the big questions were never answered.

OPEN PROBLEM 1: What are the forbidden subgraphs of the class of circle graphs? Is there a polynomial algorithm for recognizing circle graphs?

Kabell [1980] felt he had an answer to the first question. He stated that subgraphs that contain an asteroidal star, or are homeomorphic to C_5+x or $\overline{C}_6$, were the only forbidden subgraphs of the class of circle graphs. However, P_7^2+x must also be added to this list and possibly other graphs.

The Berge Strong Perfect Graph Conjecture has been fully settled with respect to circle graphs. But not for graphs in general.

OPEN PROBLEM 2: Is the Berge Stong Perfect Graph Conjecture true? Are there other classes of graphs for which it holds? What other properties do partitionable graphs have? In light of these

properties, what other proofs can be simplified, for classes of graphs for which the conjecture holds?

The complexity of calculating various parameters of a graph holds many open doors. For the class of circle graphs the maximum clique and maximum stable set problems are polynomial, and the coloring problem is NP-complete. But what about larger classes of graphs?

OPEN PROBLEM 3: What is the complexity of the minimum clique cover problem for the class of circle graphs? What is the complexity of the maximum clique, maximum stable set, and minimum clique cover problems for larger classes of graphs, e.g., the class of graphs derived from the intersecting lines in a plane?

APPENDIX A

A.1 INTRODUCTION TO THE BALM PROGRAMMING LANGUAGE

The algorithms of chapter 4 are presented here in the BALM programming language developed at New York University. BALM is a highly structured language. In general it is very easy to follow an algorithm written in BALM. Nevertheless, a few guidelines are in order.

Variables are typeless, in that their value contains the type information. Values are maintained in a heap which is garbage collected as is necessary. The data types of BALM are: logical, integer, label, pointer-to-identifier, pointer-to-code, pointer-to-string, pointer-to-pair and pointer-to-vector. In the programs of this appendix only the following data types are used: integer, pointer-to-code, pointer-to-pair and pointer-to-vector. A variable receives a value by the operator "=". The data type of integer is clear, and the data type of pointer-to-code is simply a procedure. The data types of pointer-to-pair and pointer-to-vector need a brief explanation.

Let x be a variable having a value of a pointer-to-pair. The pair of x can be separated by using hd(x) and tl(x), parentheses being optional, where hd(x) and tl(x) are valid

BALM data types. Clearly a list can be easily constructed by letting tl(x) be another pointer-to-pair. Since list processing is quite common, a function is provided to create a list of the values of a,b,...,z by x= LIST(a,b,...,z). A single pair can also be constructed by x= A:B; thus hd(x) is the value of A and tl(x) is the value of B. An explicit list can be also constructed by prefacing with "=" a parenthesized list having no commas: x= =(1 2 3). LFROMV makes a list out of the values of a vector.

The data type of pointer-to-vector is more straight forward. x= MAKVECTO(n) creates a vector of length n having undefined values; x[2] accesses the second value of the vector; x= VECTOR(a,b,...,z) creates a vector of the values of a,b,...,z; and s= =[1 2 3] explicity establishes a vector. VFROML makes a vector from the values of a list. The data types of the elements of a vector need not be homogeneous.

The basic control structures of BALM are:

1) variable = PROC (pass by value variable list), statement END
2) BEGIN (local variable list), statement list END
3) DO statement list END
4) FOR variable = (exp_1, exp_2, exp_3) REPEAT statement
5) WHILE condition REPEAT statement
6) IF condition THEN statement ELSE statement

Statement is understood in a recursive manner, allowing for nested control structures and collections of assignments. Name scoping is dynamic in that each variable can be considered as a global stack with a "push" each time it is declared in an executed BEGIN local variable list, and "popped" when the END of that BEGIN occurs.

A full detailed description of the BALM programming language can be obtained from the NYU Computer Science Department (see Harrison and Brown [1974]).

A.2 PROGRAMS TO FIND π AND s, GIVEN THE OTHER, SUCH THAT H(π) = G(s) + ISOLATED NODES

```
MBALM EXECUTION      80/08/07.

READING BLM4SVD
"BALM4.2.2"

OFROMP = PROC (P), BEGIN (N, M, O, I, P1, O1),
  N= SIZE(P), M= MAKVECTO(N), O= MAKVECTO(2*N),
  FOR I= (1, N) REPEAT M[I]= NIL,
  P1= O1= 0,
* PROCESS EACH INTERVAL IN ORDER.
  FOR I= (1, N) REPEAT DO
    COMMENT "KEEP DRAWING FROM P UNTIL I IS PASSED"
    WHILE M[I] EQ NIL REPEAT DO
      P1= P1 + 1, O1= O1 + 1, O[O1]= P[P1], M[P[P1]]= 1 END,
    O1= O1 + 1, O[O1]= I
    END,
  RETURN (O)
  END END;


DO P= =[2 9 4 6 7 1 3 8 5],
GEN= OFROMP(P),
PRINT ("THE CORRESPONDING CIRCLE GRAPH SEQUENCE OF", P, "IS",
       GEN)
```

```
END;
"THE CORRESPONDING CIRCLE GRAPH SEQUENCE OF" [2 9 4 6 7 1 3 8
 5] "IS" [2 9 4 6 7 1 1 2 3 3 4 8 5 5 6 7 8 9]


DO P= =[7 4 2 10 6 1 8 3 9 5],
GEN= OFROMP(P),
PRINT ("THE CORRESPONDING CIRCLE GRAPH SEQUENCE OF", P, "IS",
      GEN)
END;
"THE CORRESPONDING CIRCLE GRAPH SEQUENCE OF" [7 4 2 10 6 1 8
 3 9 5] "IS" [7 4 2 10 6 1 1 2 8 3 3 4 9 5 5 6 7 8 9 10]


OFROMP= NIL;


PFROMO = PROC (O),
  BEGIN (NP, N, M, OP, I, O1, ADD, L, L1, P, P1),
  COMMENT "OP WILL BE THE SEQUENCE AFTER ADDING THE
           ISOLATED INTERVALS AND NP ITS SIZE"
  NP= N= SIZE(O)/2, M= MAKVECTO(N), OP= MAKVECTO(4*N),
  FOR I= (1, N) REPEAT M[I]= NIL,
  O1= ADD= 0,
* ADD THE EXTRA INTERVALS.
  FOR I= (1, 2*N) REPEAT DO
    IF M[O[I]] EQ NIL THEN DO
      M[O[I]]= 1, ADD= 1 END
    ELSEIF ADD EQ 1 AND O[I] NE O[I-1] THEN DO
      NP= NP + 1, O1= O1 + 2,
      OP[O1]= OP[O1-1]= NP,
      ADD= 0
      END
    ELSE ADD= 0,
    O1= O1 + 1, OP[O1]= O[I]
    END,
* RENUMBER FROM THE END.
  L= MAKVECTO(NP), FOR I= (1, NP) REPEAT L[I]= NIL,
  L1= NP,
  FOR I= (2*NP, 1, -1) REPEAT
    IF L[OP[I]] EQ NIL THEN DO
      L[OP[I]]= L1, OP[I]= L1, L1= L1 - 1 END
    ELSE OP[I]= L[OP[I]],
* READ OFF THE LEFT ENDPOINTS.
  M= MAKVECTO(NP), P= MAKVECTO(NP),
  FOR I= (1, NP) REPEAT M[I]= NIL,
  P1= 0,
  FOR I= (1, 2*NP) REPEAT
    IF M[OP[I]] EQ NIL THEN DO
      M[OP[I]]= 1, P1= P1 + 1, P[P1]= OP[I] END,
* RETURN.
  RETURN (P)
  END END;
```

```
DO O= =[1 2 3 4 5 3 6 2 7 5 1 6 7 4],
GEN= PFROMO(O),
PRINT ("THE CORRESPONDING PERMUTATION OF", O, "IS", GEN) END;
"THE CORRESPONDING PERMUTATION OF" [1 2 3 4 5 3 6 2 7 5 1 6 7
 4] "IS" [7 4 2 10 6 1 8 3 9 5]


DO O= =[2 9 4 6 7 1 1 2 3 3 4 8 5 5 6 7 8 9],
GEN= PFROMO(O),
PRINT ("THE CORRESPONDING PERMUTATION OF", O, "IS", GEN) END;
"THE CORRESPONDING PERMUTATION OF" [2 9 4 6 7 1 1 2 3 3 4 8 5
 5 6 7 8 9] "IS" [2 9 4 6 7 1 3 8 5]


DO O= =[1 2 1 3 2 4 3 5 4 5],
GEN= PFROMO(O),
PRINT ("THE CORRESPONDING PERMUTATION OF", O, "IS", GEN) END;
"THE CORRESPONDING PERMUTATION OF" [1 2 1 3 2 4 3 5 4 5] "IS"
[2 4 1 6 3 8 5 9 7]


* * * (END OF FILE ON INPUT) * * *
```

A.3 PROGRAMS TO FIND THE EDGES OF G(s), C(s) and D(s)

```
MBALM EXECUTION      80/08/07.

READING BLM4SVD
"BALM4.2.2"

EDGES = PROC (O), BEGIN (N, M, I, S, C, EO, EC, ED, T, R),
  N= SIZE(O)/2, M= MAKVECTO(N),
  FOR I= (1, N) REPEAT M[I]= NIL,
  COMMENT "S AND C ARE THE STACKS OF OPEN AND CLOSED INTERVALS,
           RESPECTIVELY, AND EO, EC AND ED ARE THE LISTS OF
           EDGES FOR THE OVERLAP, CONTAINMENT AND DISJOINT
           GRAPHS, RESPECTIVELY"
  S= NIL:NIL, C= NIL:NIL, EO= EC= ED= NIL,
* PROCESS EACH ENDPOINT OF O SEQUENTIALLY.
  FOR I= (1, 2*N) REPEAT
    IF M[O[I]] EQ NIL THEN DO
      M[O[I]]= 1,
      COMMENT "ADD TO THE TOP OF S"
      TL S= O[I]:TL S,
      COMMENT "GENERATE THE ADJ OF O[I] IN ED BY SCANNING C"
```

```
      T= TL C,
      WHILE T NE NIL REPEAT DO
        ED= LIST(HD T, O[I]):ED, T= TL T END
      END
    ELSE DO
      M[O[I]]= NIL,
      COMMENT "GENERATE THE ADJ OF O[I] IN EO
               BY SCANNING THE TOP OF S"
      T= S,
      WHILE HD TL T NE O[I] REPEAT DO
        T= TL T, EO= LIST(O[I], HD T):EO END,
      COMMENT "DELETE O[I] FROM S AND ADD TO C"
      TL T= TL TL T, TL C= O[I]:TL C,
      COMMENT "GENERATE THE ADJ OF O[I] IN EC
               BY SCANNING THE BOTTOM OF S"
      T= TL T,
      WHILE T NE NIL REPEAT DO
        EC= LIST(O[I], HD T):EC, T= TL T END
      END,
  R= LIST(REVERSAL(EO), REVERSAL(EC), REVERSAL(ED)),
  RETURN (R)
  END END;


REVERSAL = PROC (L), BEGIN (R),
* REVERSES THE LIST L.
  R= NIL,
  WHILE L NE NIL REPEAT DO
    R= HD L:R, L= TL L END,
  RETURN (R)
  END END;


DO O= =[1 2 3 4 5 6 6 1 7 7 3 8 9 9 4 5 8 2],
GEN= EDGES(O),
PRINT ("THE EDGES OF", O, "ARE:"),
PRINT ("OVERLAP=", HD GEN),
PRINT ("CONTAINMENT=", HD TL GEN),
PRINT ("DISJOINT=", HD TL TL GEN) END;
"THE EDGES OF" [1 2 3 4 5 6 6 1 7 7 3 8 9 9 4 5 8 2] "ARE:"
"OVERLAP=" ((1 5) (1 4) (1 3) (1 2) (3 5) (3 4) (4 8) (4 5) (
  5 8))
"CONTAINMENT=" ((6 5) (6 4) (6 3) (6 2) (6 1) (7 5) (7 4) (7
  3) (7 2) (3 2) (9 8) (9 5) (9 4) (9 2) (4 2) (5 2) (8 2))
"DISJOINT=" ((1 7) (6 7) (3 8) (7 8) (1 8) (6 8) (3 9) (7 9)
 (1 9) (6 9))


* * * (END OF FILE ON INPUT) * * *
```

A.4 PROGRAMS TO FIND THE MAXIMUM (WEIGHTED) CLIQUE

```
MBALM EXECUTION      80/08/07.

READING BLM4SVD
"BALM4.2.2"

MAXOCLIQ = PROC (O),
  BEGIN (N, M, R, I, S, MX, MXC, P, T, J, GEN),
  N= SIZE(O)/2,
* PREPROCESS:  LOCATE THE RIGHT ENDPOINT OF EACH INTERVAL.
  M= MAKVECTO(N), R= MAKVECTO(N),
  FOR I= (1, N) REPEAT M[I]= NIL,
  FOR I= (1, 2*N) REPEAT
    IF M[O[I]] EQ NIL THEN M[O[I]]= I
    ELSE DO M[O[I]]= NIL, R[O[I]]= I END,
* RUN THE ALGORITHM.
  COMMENT "S IS THE STACK OF OPEN INTERVALS, MXC AND MX
           KEEP THE LARGEST CLIQUE AND ITS SIZE"
  S= NIL:NIL, MX= 0, MXC= NIL,
  FOR I= (1, 2*N) REPEAT
    IF M[O[I]] EQ NIL THEN DO
      M[O[I]]= I,
      COMMENT "ADD TO THE TOP OF S"
      TL S= O[I]:TL S
      END
    ELSE DO
      M[O[I]]= NIL,
      COMMENT "PICK UP THE ADJACENCY OF O[I]"
      P= NIL, T= S,
      WHILE HD TL T NE O[I] REPEAT DO
        T= TL T, P= HD T:P END,
      TL T= TL TL T, COMMENT "DELETE O[I] FROM S"
      COMMENT "FORM A VECTOR AND GENERATE THE RIGHT ENDPOINTS"
      P= VFROML(P),
      FOR J= (1, SIZE(P)) REPEAT P[J]= R[P[J]],
      COMMENT "FIND THE MAX CLIQUE"
      GEN= IF SIZE(P) EQ 0 THEN 0:NIL ELSE MAXPCLIQ(P),
      IF MX LT (HD GEN)+1 THEN DO COMMENT "PICK THE BEST"
        MX= (HD GEN)+1, MXC= I:TL GEN END
      END,
* CONVERT THE RIGHT ENDPOINTS OF MXC BACK TO INTERVALS.
  T= MXC,
  WHILE T NE NIL REPEAT DO
    HD T= O[HD T], T= TL T END,
  RETURN (MX:MXC)
  END END;


MAXPCLIQ = PROC (P), BEGIN (N, L, MX, MXC, I, A, B, C),
  COMMENT "L IS THE COLLECTION OF STACKS WITH MX THE NUMBER
```

```
          OF USED STACKS AND MXC THE RESPECTIVE CLIQUES"
  N= SIZE(P), L= MAKVECTO(N), MX= 0, MXC= MAKVECTO(N),
  FOR I= (1, N) REPEAT DO
*   STACK P[I] USING A BINARY SEARCH.
    A= 1, C= MX+1,
    WHILE A NE C REPEAT DO
      B= (A+C)/2,
      IF P[I] GT L[B] THEN A= B+1
      ELSE C= B
      END,
    L[A]= P[I],
    MXC[A]= P[I]:(IF A GT 1 THEN MXC[A-1] ELSE NIL),
    IF MX LT A THEN MX= A
    END,
  RETURN (MX:REVERSAL(MXC[MX]))
  END END;



REVERSAL = PROC (L), BEGIN (R),
* REVERSES THE LIST L.
  R= NIL,
  WHILE L NE NIL REPEAT DO
    R= HD L:R, L= TL L END,
  RETURN (R)
  END END;


DO O= =[1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7],
GEN= MAXOCLIQ),
PRINT ("THE MAXOCLIQUE OF", O, "IS OF SIZE", HD GEN, "BEING",
       TL GEN)
END;
"THE MAXOCLIQUE OF" [1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7]
"IS OF SIZE" 4 "BEING" (3 4 6 7)


* * * (END OF FILE ON INPUT) * * *

MBALM EXECUTION      80/08/07.

READING BLM4SVD
"BALM4.2.2"

MAXWOCLQ = PROC (O, W),
  BEGIN (N, M, R, I, S, MX, MXC, P, T, J, GEN),
  N= SIZE(O)/2,
* PREPROCESS:  LOCATE THE RIGHT ENDPOINT OF EACH INTERVAL.
  M= MAKVECTO(N), R= MAKVECTO(N),
  FOR I= (1, N) REPEAT M[I]= NIL,
  FOR I= (1, 2*N) REPEAT
    IF M[O[I]] EQ NIL THEN M[O[I]]= 1
    ELSE DO M[O[I]]= NIL, R[O[I]]= I END,
```

```
* RUN THE ALGORITHM.
  COMMENT "S IS THE STACK OF OPEN INTERVALS, MXC AND
           MX KEEP THE LARGEST CLIQUE AND ITS SIZE"
  S= NIL:NIL, MX= 0, MXC= NIL,
  FOR I= (1, 2*N) REPEAT
    IF M[O[I]] EQ NIL THEN DO
      M[O[I]]= 1,
      COMMENT "ADD TO THE TOP OF S"
      TL S= O[I]:TL S
      END
    ELSE DO
      M[O[I]]= NIL,
      COMMENT "PICK UP THE ADJACENCY OF O[I]"
      P= NIL, T= S,
      WHILE HD TL T NE O[I] REPEAT DO
        T= TL T, P= HD T:P END,
      TL T= TL TL T, COMMENT "DELETE O[I] FROM S"
      COMMENT "FORM A VECTOR AND GENERATE THE RIGHT ENDPOINTS"
      P= VFROML(P),
      FOR J= (1, SIZE(P)) REPEAT P[J]= R[P[J]],
      COMMENT "FIND THE MAX CLIQUE"
      GEN= IF SIZE(P) EQ 0 THEN 0:NIL ELSE MAXWPCLQ(P, O, W),
      COMMENT "PICK THE BEST"
      IF MX LT (HD GEN)+W[O[I]] THEN DO
        MX= (HD GEN)+W[O[I]], MXC= I:TL GEN END
      END,
* CONVERT THE RIGHT ENDPOINTS OF MXC BACK TO INTERVALS.
  T= MXC,
  WHILE T NE NIL REPEAT DO
    HD T= O[HD T], T= TL T END,
  RETURN (MX:MXC)
  END END;


MAXWPCLQ = PROC (P, O, W), BEGIN (T, MX, MXC),
* MAKE P INTO A LIST AND TAG EACH ELEMENT WITH
* THE WEIGHT AND CLIQUE IN WHICH IT IS CONTAINED.
  P= NIL:LFROMV(P), T= TL P,
  WHILE T NE NIL REPEAT DO
    HD T= VECTOR(HD T, W[O[HD T]], HD T:NIL), T= TL T END,
  COMMENT "MCX AND MX KEEP THE LARGEST CLIQUE AND ITS SIZE"
  MX= 0, MXC= NIL,
* PROCESS EACH ELEMENT.
  COMMENT "O, W, MX, MXC ARE GLOBAL TO SUBMAXWP"
  WHILE TL P NE NIL REPEAT SUBMAXWP (P),
  RETURN (MX:REVERSAL(MXC))
  END END;

SUBMAXWP = PROC (P),
  COMMENT "O, W, MX, MXC ARE GLOBAL FROM MAXWPCLQ"
  BEGIN (PP, T, TT),
  PP= HD TL P, T= TL P,
* COMPARE THE HEAD OF THE LIST WITH THE TAIL.
```

```
  WHILE TL T NE NIL REPEAT DO
    TT= HD TL T,
    IF PP[1] LT TT[1] THEN DO
      COMMENT "HEAD IS LESS, CHECK FOR CLIQUE IMPROVEMENT"
      IF TT[2] LT PP[2]+W[O[TT[1]]] THEN DO
        TT[2]= PP[2]+W[O[TT[1]]], TL TT[3]= PP[3] END,
      T= TL T
      END
    ELSE
      COMMENT "TAIL ELEMENT IS LESS, PROCESS IT OUT BEFORE
               CONTINUING"
      SUBMAXWP (T)
    END,
* CHOOSE THE BEST AND DELETE THE HEAD.
  IF MX LT PP[2] THEN DO MX= PP[2], MXC= PP[3] END,
  TL P = TL TL P,
  RETURN (NIL)
  END END;


REVERSAL = PROC (L), BEGIN (R),
* REVERSES THE LIST L.
  R= NIL,
  WHILE L NE NIL REPEAT DO
    R= HD L: R, L= TL L END,
  RETURN (R)
  END END;


DO O= =[1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7],
W= =[1 1 1 1 1 1 1 1],
GEN= MAXWOCLQ(O, W),
PRINT ("THE MAXWOCLIQUE OF", O, "WITH WEIGHTS", W,
       "IS OF SIZE", HD GEN, "BEING", TL GEN) END;
"THE MAXWOCLIQUE OF" [1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7]
"WITH WEIGHTS" [1 1 1 1 1 1 1 1] "IS OF SIZE" 4 "BEING"
(3 4 6 7)


DO O= =[1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7],
W= =[4 1 1 1 1 1 1 1],
GEN= MAXWOCLQ(O, W),
PRINT ("THE MAXWOCLIQUE OF", O, "WITH WEIGHTS", W,
       "IS OF SIZE", HD GEN, "BEING", TL GEN) END;
"THE MAXWOCLIQUE OF" [1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7]
"WITH WEIGHTS" [4 1 1 1 1 1 1 1] "IS OF SIZE" 5 "BEING" (1 7)


* * * (END OF FILE ON INPUT) * * *
```

A.5 PROGRAMS TO FIND THE MAXIMUM (WEIGHTED) STABLE SET

```
MBALM EXECUTION      80/08/07.

READING BLM4SVD
"BALM4.2.2"

MAXWOSTB = PROC (O, W),
  BEGIN (N, M, T, L, ADJ, V, TT, CL, I, C, GEN),
  N= SIZE(W),
* BUILD THE CONTAINMENT ADJACENCIES.
  T= L= NIL:LFROMV(O), M= MAKVECTO(N), ADJ= MAKVECTO(N),
  FOR I= (1, N) REPEAT M[I]= NIL,
  COMMENT "RUN DOWN L"
  WHILE TL T NE NIL REPEAT DO
    V= HD TL T,
    IF M[V] EQ NIL THEN DO
      M[V]= 1, T= TL T END
    ELSE DO
      M[V]= NIL,
      COMMENT "PICK UP ADJ[V]"
      TT= L, ADJ[V]= NIL,
      WHILE (HD TL TT) NE V REPEAT DO
        TT= TL TT, ADJ[V]= HD TT:ADJ[V] END,
      COMMENT "DELETE BOTH OCCURRENCES OF V FROM L"
      TL T= TL TL T, IF T EQ TL TT THEN T= TT,
      TL TT= TL TL TT
      END
    END,
  M= T= L=NIL,
* BUILD THE CONTAINMENT LISTS.
  CL= MAKVECTO(N), FOR I= (1, N) REPEAT CL[I]= NIL,
  FOR I= (1, 2*N) REPEAT DO
    COMMENT "ADD O[I] TO EACH CL OF ITS ADJACENCY"
    T= ADJ[O[I]],
    WHILE T NE NIL REPEAT DO
      CL[HD T]= O[I]:CL[HD T], T= TL T END
    END,
  ADJ= NIL,
  FOR I= (1, N) REPEAT CL[I]= VFROML(REVERSAL(CL[I])),
* RUN THE ALGORITHM.
  C= MAKVECTO(N), FOR I= (1, N) REPEAT C[I]= NIL,
  GEN= SUBMAXWO(O), COMMENT "W, CL, C ARE GLOBAL TO SUBMAXWO"
  TL GEN= REVERSAL(UNTANGLE(TL GEN)),
  RETURN (GEN)
  END END;


SUBMAXWO = PROC (O),
  COMMENT "W, CL, C ARE GLOBAL FROM MAXWOSTB"
  BEGIN (N, MX, MXS, NN, I, L, NW),
```

```
  COMMENT "AS O IS FOLLOWED MXS AND MX KEEP THE LARGEST
           WEIGHTED STABLE SET AND ITS WEIGHT AT EACH POSITION"
  N= SIZE(O)/2, MX= MAKVECTO(2*N), MXS= MAKVECTO(2*N),
  NN= O[1],
  FOR I= (1, 2*N) REPEAT IF NN LT O[I] THEN NN= O[I],
  L= MAKVECTO(NN), FOR I= (1, 2*N) REPEAT L[O[I]]= NIL,
  FOR I= (1, 2*N) REPEAT
    IF L[O[I]] EQ NIL THEN DO
      L[O[I]]= I,
      MX[I]= IF I EQ 1 THEN 0 ELSE MX[I-1],
      MXS[I]= IF I EQ 1 THEN NIL ELSE MXS[I-1]
      END
    ELSE DO
      IF C[O[I]] EQ NIL THEN
        C[O[I]]= IF SIZE(CL[O[I]]) EQ 0 THEN 0:NIL
                 ELSE SUBMAXWO(CL[O[I]]),
      NW= HD(C[O[I]]) + W[O[I]] + MX[L[O[I]]],
      COMMENT "PICK THE BEST"
      IF NW LE MX[I-1] THEN DO
        MX[I]= MX[I-1], MXS[I]= MXS[I-1] END
      ELSE DO
        MX[I]= NW,
        MXS[I]= LIST(MXS[L[O[I]]], O[I], TL C[O[I]])
        END
      END,
  RETURN (LIST(MX[2*N], MXS[2*N]))
  END END;


UNTANGLE = PROC (L), BEGIN (R),
* MAKES THE TREE L INTO A LIST R.
  R= NIL,
  SUBUNTGL (L), COMMENT "R IS GLOBAL TO SUBUNTGL"
  RETURN (R)
  END END;

SUBUNTGL = PROC (L), COMMENT "R IS GLOBAL FROM UNTANGLE
  IF INTQ(L) THEN R=L:R
  ELSEIF PAIRQ(L) THEN DO
    SUBUNTGL (HD L), SUBUNTGL (TL L) ENN
  END;


REVERSAL = PROC (L), BEGIN (R),
* REVERSES THE LIST L.
  R= NIL,
  WHILE L NE NIL REPEAT DO
    R= HD L:R, L= TL L END,
  RETURN (R)
  END END;


DO
```

```
O= =[1 2 3 4 5 3 4 5 6 2 6 1],
W= =[1 1 1 1 1 1],
GEN= MAXWOSTB(O, W),
PRINT ("THE MAXWOSTABLE OF", O, "WITH WEIGHTS", W,
       "IS OF WEIGHT", HD GEN, "BEING", TL GEN)
END;
"THE MAXWOSTABLE OF" [1 2 3 4 5 3 4 5 6 2 6 1] "WITH WEIGHTS"
[1 1 1 1 1 1] "IS OF WEIGHT" 3 "BEING" (1 2 3)


DO
O= =[1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7],
W= =[1 1 1 1 1 1 1 1],
GEN= MAXWOSTB(O, W),
PRINT ("THE MAXWOSTABLE OF", O, "WITH WEIGHTS", W,
       "IS OF WEIGHT", HD GEN, "BEING", TL GEN)
END;
"THE MAXWOSTABLE OF" [1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7]
"WITH WEIGHTS" [1 1 1 1 1 1 1 1] "IS OF WEIGHT" 5 "BEING"
(1 2 5 6 8)


DO
O= =[1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7],
W= =[1 1 1 1 1 1 3 1],
GEN= MAXWOSTB(O, W),
PRINT ("THE MAXWOSTABLE OF", O, "WITH WEIGHTS", W,
       "IS OF WEIGHT", HD GEN, "BEING", TL GEN)
END;
"THE MAXWOSTABLE OF" [1 2 3 4 5 5 2 6 7 8 8 3 4 6 1 7]
"WITH WEIGHTS" [1 1 1 1 1 1 3 1] "IS OF WEIGHT" 6 "BEING"
(2 5 7 8)


* * * (END OF FILE ON INPUT) * * *
```

BIBLIOGRAPHY

C. Berge, "Färbung von Graphen, deren sämtliche bzw. deren ungerade Kreise starr sind", (Zusammenfassung), Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe (1961) 114-115.

C. Berge, "Sur une conjecture relative au problème des codes optimaux", Commum. 13ième Assemblée Générale de l'URSI, Tokyo (1962).

R.G. Bland, H.-C. Huang and L.E. Trotter, Jr., "Graphical Properties Related to Minimal Imperfection", Discrete Mathematics 27 (1979) 11-22.

V. Chvátal, "On the Strong Perfect Graph Conjecture", Journal of Combinatorial Theory (B) 20 (1976) 139-141.

V. Chvátal, R.L. Graham, A.F. Perold and S.H. Whitesides, "Combinatorial Designs Related to the Strong Perfect Graph Conjecture", Discrete Mathematics 26 (1979) 83-92.

G. Ehrlich, S. Even and R.E. Tarjan, "Intersection Graphs of Curves in the Plane", Journal of Combinatorial Theory (B) 21 (1976) 8-20.

S. Even and A. Itai, "Queues, Stacks and Graphs", Theory of Machines and Computations (Z. Kohavi and A. Paz, editors, Academic Press, New York, NY, 1971) 71-86.

S. Even, A. Pnueli and A. Lempel, "Permutation Graphs and Transitive Graphs", Journal of the Association for Computing Machinery 19 (1972) 400-410.

M.L. Fredman, "On Computing the Length of Largest Increasing Subsequences", Discrete Mathematics 11 (1975) 29-35.

M.R. Garey, D.S. Johnson, G.L. Miller and C.H. Papadimitriou, "The Complexity of Coloring Circular Arcs and Chords", Report, Bell Laboratories and M.I.T. (1979), submitted for publication.

F. Gavril, "Algorithms for a Maximum Clique and a Maximum Independent Set of a Circle Graph", Networks 3 (1973) 261-273.

M.C. Golumbic, Algorithmic Graph Theory and Perfect Graphs (Academic Press, New York, NY, 1980).

C.M. Grinstead, "Toroidal Graphs and the Strong Perfect Graph Conjecture", Ph.D. Thesis, University of California at Los Angeles (1978).

M.C. Harrison and S. Brown, The BALM Programming Language (Courant Institute of Mathematical Sciences, New York University, New York, NY, 1974).

J.A. Kabell, "Intersection Graphs: Structure and Invariants", Ph.D. Thesis, Department of Mathematics, University of Michigan (1980).

C.G. Lekkerkerker and J.Ch. Boland, "Representation of a Finite Graph by a Set of Intervals on the Real Line", Fundamenta Mathematicae 51 (1962) 45-64.

L. Lovász, "Normal Hypergraphs and the Perfect Graph Conjecture", Discrete Mathematics 2 (1972) 253-267.

L. Lovász, "A Characterization of Perfect Graphs", Journal of Combinatorial Theory (B) 13 (1972) 95-98.

M.W. Padberg, "Perfect Zero-One Matrices", Mathematical Programming 6 (1974) 180-196.

M.W. Padberg, "Almost Integral Polyhedra Related to Certain Combinatorial Optimization Problems", Linear Algebra and Its Applications 15 (1976) 69-88.

K.R. Parthasarathy and G. Ravindra, "The Strong Perfect Graph Conjecture is True for $K_{1,3}$-free Graphs", Journal of Combinatorial Theory (B) 21 (1976) 212-223.

K.R. Parthasarathy and G. Ravindra, "The Validity of the Strong Perfect-Graph Conjecture for (K_4-e)-free Graphs", Journal of Combinatorial Theory (B) 26 (1979) 98-100.

H. Sachs, "On the Berge Conjecture Concerning Perfect Graphs", Combinatorial Structures and Their Applications (R. Guy, et. al., editors, Gordon and Breach, New York, 1970) 377-384.

R. Tarjan, "Sorting Using Networks of Queues and Stacks", Journal of the Association for Computing Machinery 19 (1972) 341-346.

A. Tucker, "The Strong Perfect Graph Conjecture for Planar Graphs", Canadian Journal of Mathematics 25 (1973) 103-114.

A. Tucker, "Coloring a Family of Circular Arcs", SIAM Journal of Applied Mathematics 29 (1975) 493-502.

A. Tucker, "Critical Perfect Graphs and Perfect 3-Chromatic Graphs", Journal of Combinatorial Theory (B) 23 (1977) 143-149.

This book may be kept

FOURTEEN DAYS

A fine will be charged for each day the book is kept overtime.

GAYLORD 142 PRINTED IN U.S.A.

NYU NSO-21 c.2
Buckingham
Circle graphs.

N.Y.U. Courant Institute of
Mathematical Sciences
251 Mercer St.
New York, N. Y. 10012